1974

A comparison of rate 1/n convolutional codes over a simulated noisy channel using the Viterbi decoding algorithm

Haney, James Howard

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/16928

A COMPARISON OF RATE 1/n CONVOLUTIONAL CODES OVER A SIMULATED NOISY CHANNEL USING THE VITERBI DECODING ALGORITHM

James Howard Haney

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A COMPARISON OF RATE 1/n CONVOLUTIONAL
CODES OVER A SIMULATED NOISY CHANNEL
USING THE VITERBI DECODING ALGORITHM

by

James Howard Haney

December 1974

Thesis Advisor: G. H. Marmont

Approved for public release; distribution unlimited.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Comparison of Rate 1/n Convolutional Codes Over a Simulated Noisy Channel Using The Viterbi Decoding Algorithm		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis December 1974
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) James Howard Haney		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE December 1974
		13. NUMBER OF PAGES 99
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Error Correction Code
Convolutional Code
Viterbi decoding algorithm

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The application of error detection and correction codes has advanced markedly with the advent of digital technology. However, the strides made towards employing the techniques, encoding and decoding, properly have been rather limited by the Naval forces in the United States. This paper develops a computer program, which if utilized properly, would aid in deciding what error correcting scheme is best suited for a specific channel.

20. The results obtained from testing a rate 1/n convolutional code, over a simulated channel, using a Viterbi decoder shows that this is an effective analysis procedure. Though the test runs were lengthy, much of the time required was for noise simulation. This would not be a factor if actual channel noise recordings had been available.

A Comparison of Rate 1/n Convolutional
Codes Over a Simulated Noisy Channel
Using the Viterbi Decoding Algorithm

by

James Howard Haney
Captain, United States Marine Corps
B.S., Virginia Military Institute, 1968

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1974

ABSTRACT

The application of error detection and correction codes has advanced markedly with the advent of digital technology. However, the strides made towards employing the techniques, encoding and decoding, properly have been rather limited by the Naval forces in the United States. This paper develops a computer program, which if utilized properly, would aid in deciding what error correcting scheme is best suited for a specific channel.

The results obtained from testing a rate 1/n convolutional code, over a simulated channel, using a Viterbi decoder shows that this is an effective analysis procedure. Though the test runs were lengthy, much of the time required was for noise simulation. This would not be a factor if actual channel noise recordings had been available.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

The distortion of a digital transmission, caused by noise, may require that techniques be used allowing for the detection and eventual correction of errors. In digital communications, the most significant performance parameter from either a bit or message standpoint is that of probability of error. A means of minimizing the probability of error is therefore essential to effective communication.

Error detection and correction was given great impetus by Shannon's paper of 1948, [Ref. 7], which extended the promise of reliable recovery of digital data perturbed by noise (Shannon's Second Theorem). The noisy coding theorem provides that messages can be transmitted with arbitrarily small error if the source rate is no more than the channel capacity.[1] While this sets a goal and the conditions necessary to attain it, the precise method for obtaining a specific scheme was not set forth. However, Shannon did use a random coding scheme in proving this theorem. As was shown by Rice in 1950, [Ref. 6], following Shannon's example, choosing codewords randomly leads to the result that as the code becomes very long, channel capacity is approached and

---

[1] Channel capacity is the maximum average information input to a channel, which means it is properly matched to the channel less the average uncertainty at the receiver due to channel noise.

the probability of error can be made to decrease exponentially with code length. But choosing codewords randomly is not a practical scheme.

Until the 1960's the application of channel coding theory was slow in its development. The establishment of digital circuit technology and the realization of a theoretical channel (satellite communication; AWGN) provided a powerful stimulus for the utilization of practical error detection and correction techniques. The technological improvements have continued until today fairly large memories are held by small chips, and the components needed to implement powerful coding schemes are available in reduced sizes and at reasonable prices.

The objective of this paper is to demonstrate the benefits derived from a minicomputer (DEC PDP-11/40) simulation of channel noise applied to a specific error detection and correction scheme. This simulation allows for the determination of a probability of bit error for a specifiable noise density. Results from the simulation are then used to determine statistical trends of a particular error correction technique for various internal coding parameters.

The error detection and correction scheme chosen by the author is the Viterbi decoding, [Ref.3], algorithm on a rate 1/n convolutional code. Both the encoding and decoding technique are of current interest in extraterrestrial communications, with the Viterbi algorithm being principally considered for satellite communications. Therefore, in

order to provide a complete presentation, the scope of this thesis will encompass the basic principles of convolutional encoding and Viterbi decoding, as well as the structure of the simulation program, results and conclusions.

All programming was accomplished on the DEC PDP-11/40 in machine language, thus presenting the opportunity to gain valuable insight into the computer structure while completing this thesis.

II. BACKGROUND

Digital communication systems (figure 1) are usually designed to minimize the probability of error of the received data (bits), introduced by noise (gaussian, burst, fading, multipath, etc.). Thus, error detection and correction may be varied according to the degree and type of noise. There are two common forms of error protection: (1) retransmission and (2) controlled redundancy. The first may be applied to systems which are not critically affected by the ensuing delay, whereas the second does not require significant delays. This thesis deals with the latter in an attempt to gain statistical knowledge of the particular coding scheme (controlled redundancy) under study.

Controlled redundancy techniques are commonly divided into two groups: (1) block codes and (2) convolutional (tree) codes. Again, owing to the characteristics of the channel, one of these two codes may be chosen, along with a suitable decoding scheme, to achieve the desired probability of error. A decoding scheme for block codes is batch oriented, as is its encoding, using definite algebraic operations related to the segmented structure of the code. On the other hand, convolutional codes are decoded by a statistical procedure due to its continuous (bit by bit) nature.

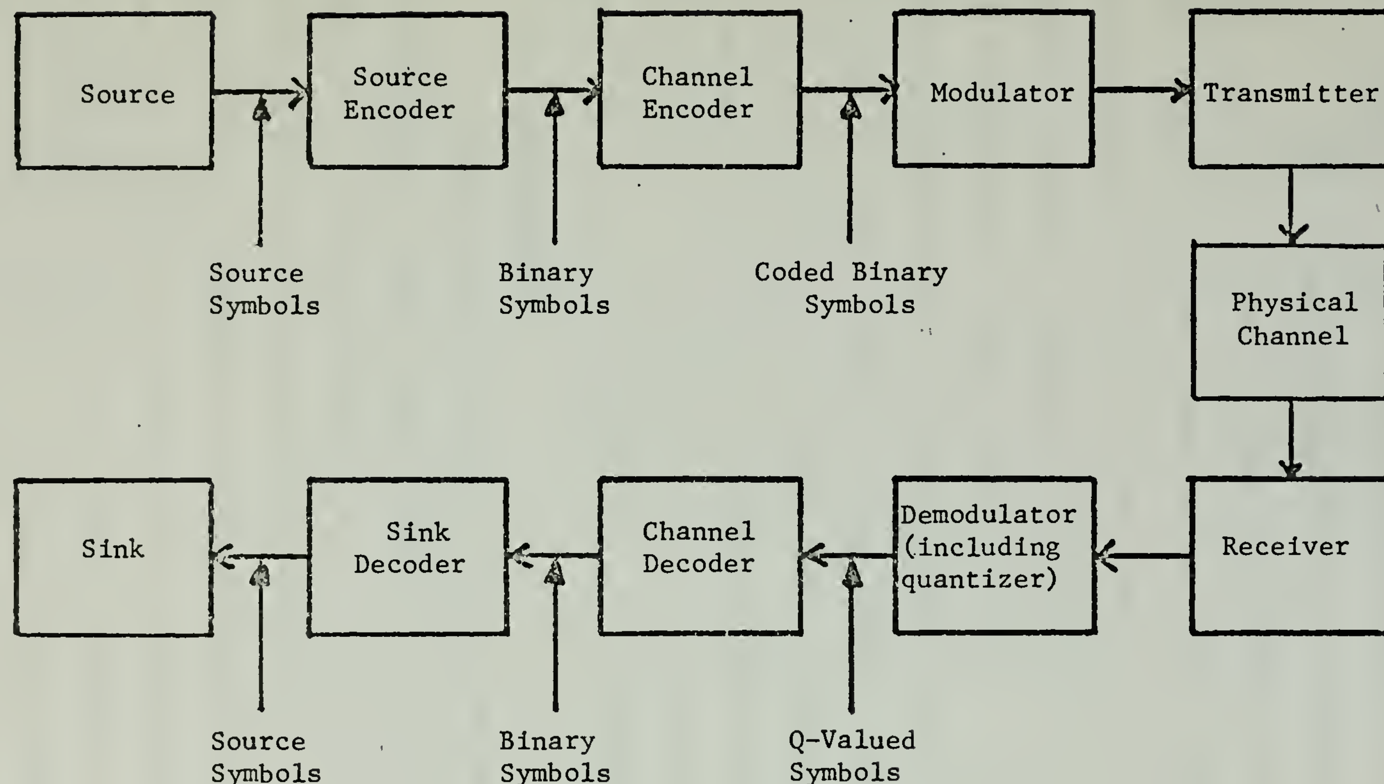


Figure 1. Model of a One-Way Digital Communication System.

A. PRINCIPLES OF CONVOLUTIONAL ENCODING

In 1955, P. Elias first proposed the use of convolutional (tree) codes for the discrete time memoryless channel, [Ref 2]. This technique extended the promise of providing a class of codes (linear) whose performance would prove superior to that of block codes of the same length, [Ref. 9]. The development of these codes also gave promise of providing for a decoder complexity increasing no more than linearly with block length and/or encoder memory. These conjectures have been verified, for the most part, by such contributors as Fano, Reiffen, Forney and Berlekamp. [Ref. 4]

Since convolutional codes form a definite discipline in coding theory, there are some aspects that will require clarification in conjunction with the implementation of the code. The next few subsections will accomplish this task.

1. Description

The process of block encoding segments (blocks) of an indefinite length input sequence into code blocks proceeds in a discrete manner, i.e., the contents of one block has no bearing on the encoding of another block. The code blocks may be completely changed from the input block or they may include the input block with check bits determined by a definite algorithm. However, when an input sequence is processed through a convolutional encoder the input/output is continuous (sequential bit to bit dependence in the encoder) and the output is generated at a specifiable number of code bits for each encoder input bit.

The term "convolutional" originates from the observation that the encoded sequence can be regarded as a convolution[2] of the input sequence with related generator sequences. An example of such a procedure is the binary convolution (denoted by *) of two sequences, $\underline{x}$ and $\underline{g}$, where $\oplus$ and $\cdot$ are binary addition and multiplication, respectively.

Input sequence:

$$\underline{x}(t_0) = (x_0 x_1 x_2 x_3 x_4 \ldots) = (1\ 0\ 1\ 1\ 0\ \ldots) ;$$

Generator sequence:

$$\underline{g}(t_0) = (g_0 g_1 g_2 g_3 g_4 \ldots) = (1\ 0\ 0\ 1\ 1\ \ldots) ,$$

where all bits in $\underline{x}$ and $\underline{g}$ are zero for x_{-i} and g_{-i}. Then the output (code) bit formed by the convolution of $x(t_0)$ with $g(t_0)$ is

$$\underline{y}(t_0) = \underline{x}(t_0) * \underline{g}(t_0) = 0 .$$

For time $t_0 + \Delta t$, the sequences and output are

$$\underline{x}(t_0+\Delta t) = (x_0 x_1 x_2 x_3 x_4 \ldots) = (1\ 0\ 1\ 1\ 0\ \ldots),$$

$$\underline{g}(t_0+\Delta t) = (g_0 g_1 g_2 g_3 g_4 \ldots) = (1\ 0\ 0\ 1\ 1\ \ldots),$$

$$y(t_0+\Delta t) = (x_0 \cdot g_0) = (1 \cdot 1) = 1 .$$

For time $t_0 + 2\Delta t$, the sequences and output are

---

[2]The convolution of two functions, f_1 and f_2, is equal to

$$\int_{-\infty}^{\infty} f_1(t-\tau) f_2(\tau) d\tau ,$$

which means one function is folded in time, then their product is integrated as one is shifted pass the other.

$$\underline{x}(t_0+2\Delta t) = (x_0 x_1 x_2 x_3 x_4 \ldots) = (1\ 0\ 1\ 1\ 0\ \ldots),$$

$$\underline{g}(t_0+2\Delta t) = (g_0 g_1 g_2 g_3 g_4 \ldots) = (1\ 0\ 0\ 1\ 1\ \ldots),$$

$$y(t_0+2\Delta t) = (x_0 \cdot g_1) \oplus (x_1 \cdot g_0) = 0 .$$

And the same procedure is applied to subsequent increments of Δt (one bit shift).

This technique extends the concept of block encoding to permit memory from bit to bit continuously as against memory within the block. Using the operation shown above, a conventional convolutional encoder (figure 2) may be defined as a linear sequential machine with k-inputs and n-outputs, where $n > k$, usually. This machine is constant, linear, causal, and finite state with operations over a finite field F, commonly binary.

The parameter k is the number of input bits which enter the encoder in a time increment (Δt). These bits along with others retained in the encoder memory (finite length), form n-output bits (code bits). For the remainder of this paper k will have a value of 1, but it should be remembered that this parameter can take on other values, as can be seen in figure 3.

The definition specified a finite state machine, whereas the example dealt with indefinite length sequences. Since an infinite length memory is not practical it becomes necessary to limit the basic concept (generator sequence) and define some new expressions employed in the actual encoder configuration.

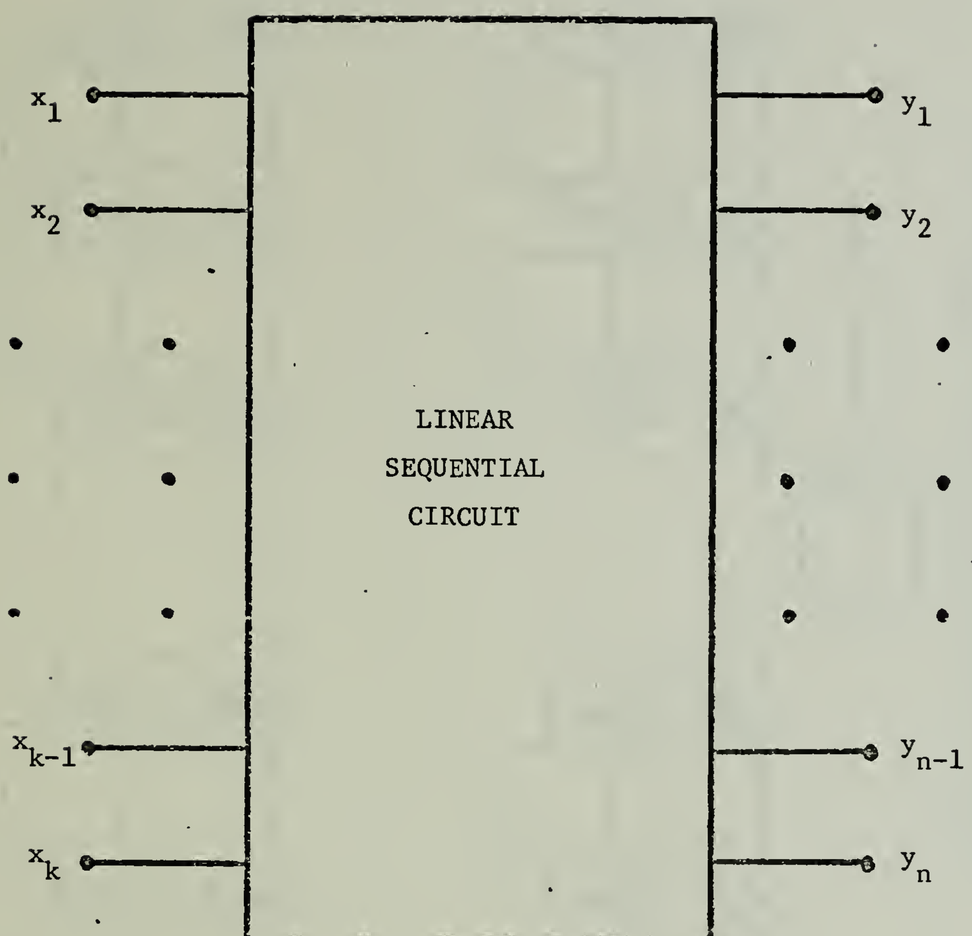


Figure 2. Conventional (n,k) Convolutional Encoder

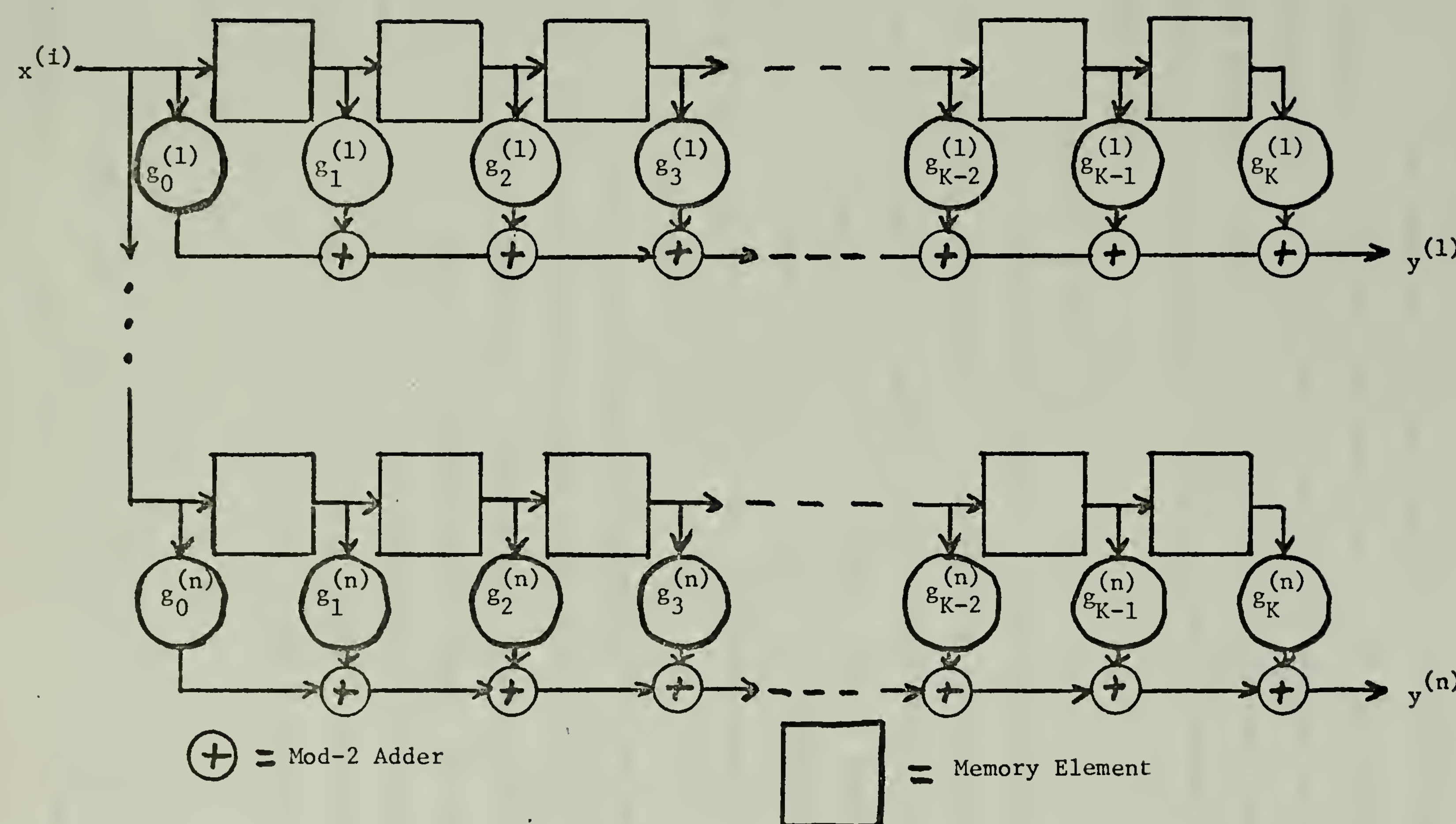


Figure 3. Representation of the i^{th} Input Section of a Rate k/n convolutional Encoder $(1 \le i \le k)$.

a. Code Rate (k/n)

The code rate is an expression of the number of k-input (1) bits per n-output bits, that is, k/n (1/n). This value may be considered a very close approximation to the actual value. The actual value would also include a very small number of zero code bits used to terminate the code sequence.

b. Memory Length (M)

For a practical encoder $\underline{g}$ must be limited in length. The memory length is a measure of this length and is the minimum number of memory cells required to generate a code. A representation of an encoder, with a memory length of 2, is shown in figure 4. This configuration corresponds to the Mealy machine in automata theory.

c. Constraint Length (K)

A more conventional expression of the length of time which an input bit affects the output sequence is that of constraint length. This term is simply the memory length plus one, and is depicted as the number of memory elements in a Moore machine configuration of an encoder (figure 5). As may be expected this value is one factor in determining the complexity of the error detection and correction scheme.

d. Free Distance (d_f)

The term distance, as applied to coding, refers to the number of differing bits between two code sequences of the same length, as seen below:

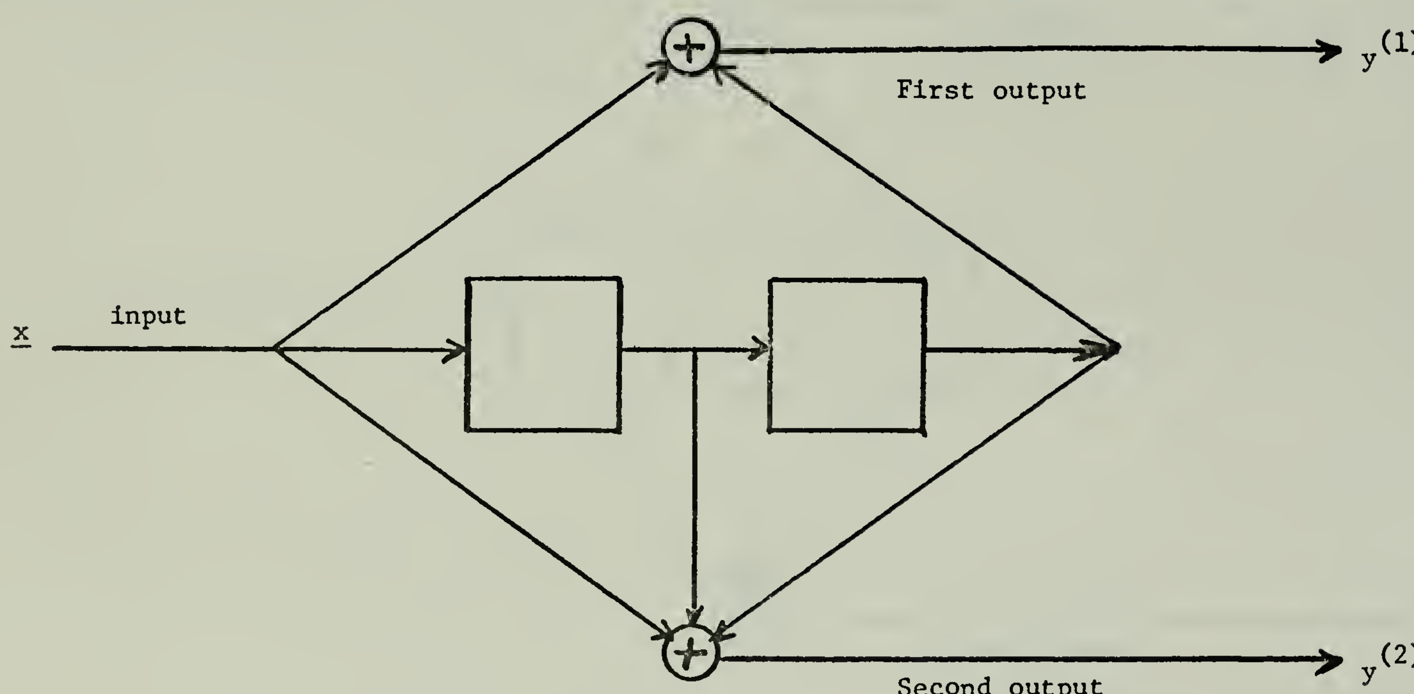


Figure 4. Mealy Configuration of a $M = 2$, $n = 2$, Binary Convolutional Encoder

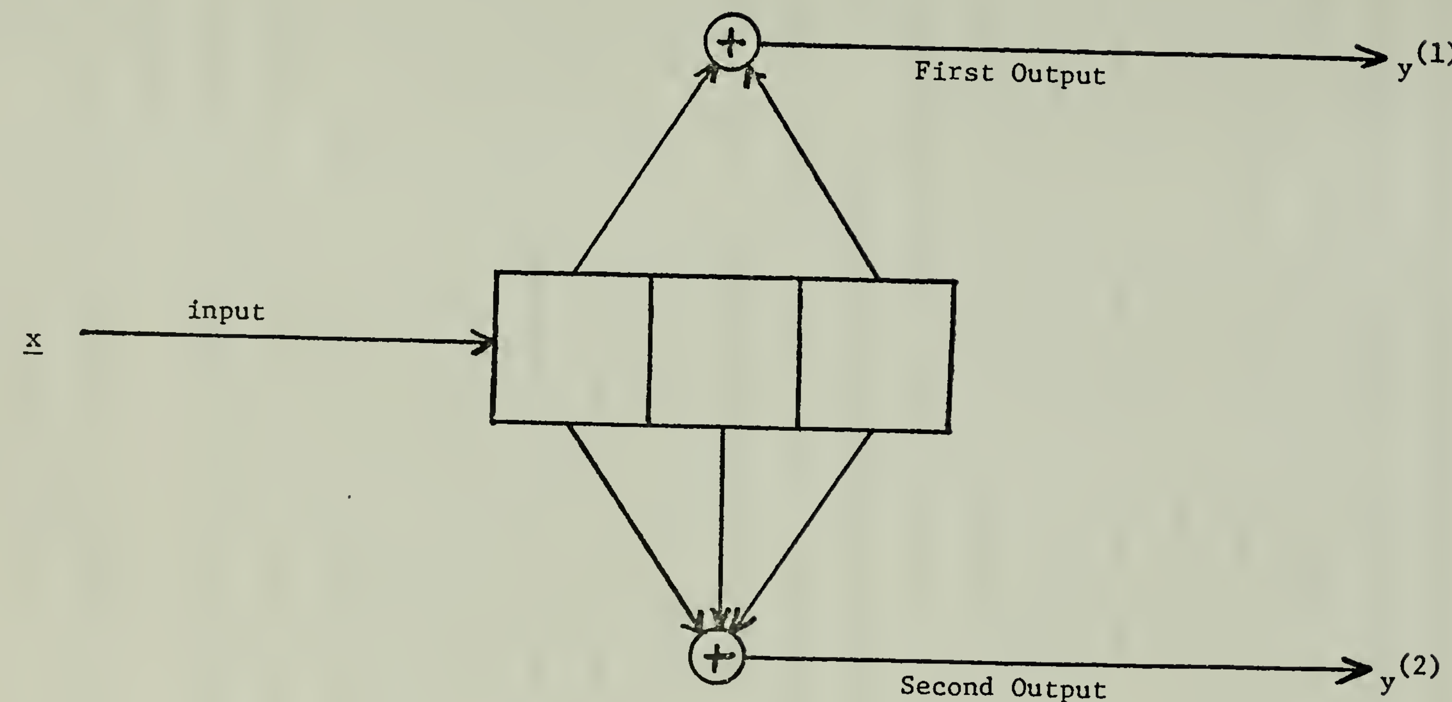


Figure 5. Moore Configuration of a K = 3, n = 2, Binary Convolutional Encoder.

code sequence 1: (1 0 0 1 0 1 1 0 0 1)

code sequence 2: (1 0 1 1 0 0 1 0 0 1)

Differences between the two sequences exist in the third and sixth bits only, therefore the distance between the two sequences is 2. When using block codes to construct a code sequence the minimum distance between all pairs of codewords (code blocks) is a definite indication of the error detecting and correcting capability of the code. However, if the code does not employ a block configuration, the use of this value may not be justified to indicate the codes error detecting and correcting capabilities.

This is the case when discussing convolutional codes. Free distance is the term for the value expressing the theoretical error correcting capability of a convolutional code. This value is defined as that minimum number of set bits occurring in a code sequence which resulted from the input of a nonzero sequence. Table I shows how the free distance (d_f) can be determined for a rate 1/2 convolutional code with $K=3$. As can be seen, the larger the value of K then the number of possible input sequence to be considered also increases at a great rate. But with the aid of computers the tedious grinding process is accomplished rather quickly.

The convolutional encoder is constructed to take full advantage of the free distance applied to a specific code rate and constraint length. This is accomplished through the proper selection of generator sequences. However, it

$\underline{x}$	$\underline{y}$	d
100	11 01 11	5
1100	11 10 10 11	6
10100	11 01 00 01 11	6
11100	11 10 01 10 11	7
	and so on	

Minimum $d = d_{free} = 5$

Table I. Determination of free distance for Encoder in Figure 5.

should be noted that with the computers currently available the drudgery can be passed on to the machine.

2. Properties

The properties of the encoder, as related to the generator sequence, are discussed in this section. For convenience, the set of generator sequences $(\underline{g}_1,\ldots,\underline{g}_n)$ used to generate the convolutional code is denoted by $[\underline{G}]$, a matrix representation of the encoder discussed in the next section. Figure 6 is useful in relating the following discussion to a physical communication system.

a. Property 1

Foremost, the useful encoder should generate a code which will yield the fewest number of errors in the codeword estimator. The estimator is a demodulation scheme at the receiver determining as accurately as possible the received sequence before actual decoding takes place. Along with this error minimization, the complexity of the estimator requires minimization to develop a useful error correction system. Error minimization and complexity minimization together may require some compromise in the practical system depending on the desired reception quality and system environment.

b. Property 2

The encoder $[\underline{G}]$ must have an inverse relation for decoding purposes. This is realized by the relation

$$\underline{y}[\tilde{\underline{G}}]^{-1} = \underline{x}[\underline{G}][\tilde{\underline{G}}]^{-1} = D^p\underline{x}$$

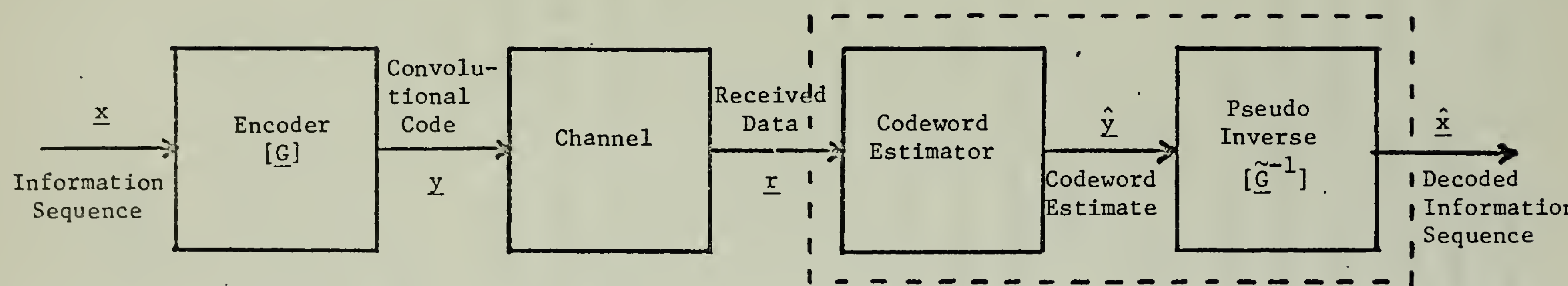


Figure 6. Block Diagram of Basic Communication System

for all $\underline{x}$. The matrix (decoder), $[\tilde{\underline{G}}]^{-1}$ is physically realizable and is a pseudo inverse, in that, the expression $[\underline{G}][\tilde{\underline{G}}]^{-1}$, yields a delay D^p, where p is the number of time intervals before decoding occurs in the expression. If p is zero the decoding occurs on receipt of the sequence and the above relation becomes

$$\underline{y}[\underline{G}]^{-1} = \underline{x}[\underline{G}][\underline{G}]^{-1} = \underline{x},$$

where $[\underline{G}]^{-1} = [\tilde{\underline{G}}]^{-1}$ when $p = 0$.

A mention of the possibility of catastrophic error propagation, [Ref], is required at this time to stress the point that $[\tilde{\underline{G}}]^{-1}$ must be feedback-free. This may be interpreted as meaning that the n generator sequences, in polynomial form should not have a common factor. If a common factor exists then the input of a sequence with a finite number of set bits to the encoder, may be decoded, after noise is added, as a sequence with an infinite number of set bits. This is catastrophic error propagation and is avoided by making $[\underline{G}]$, thus $[\tilde{\underline{G}}]^{-1}$ feedback-free.

c. Property 3

The matrix $[\underline{G}]$ must also meet the requirements mentioned in the definition of a convolutional encoder.

(1) Constant (time invariant). The time invariance of the encoder is represented by

$$[\underline{G}](D^p\underline{x}) = D^p\underline{y},$$

which means if all the inputs are shifted in time, then all the outputs are shifted accordingly.

(2) Linear. The output sequence resulting from the superposition of two input sequences, $\underline{x}$, must be equal to the superposition of the two output (code) sequences, $\underline{y}$, that would result from the inputs entered separately. This is also necessary for the multiplication of an input by a scalar as shown below. If $G:\underline{x}\rightarrow\underline{y}$, then $G(\underline{x}_1+\underline{x}_2) = G(\underline{x}_1) + G(\underline{x}_2) = \underline{y}_1 + \underline{y}_2$, and $G(\alpha\underline{x}_1) = \alpha G(\underline{x}_1) = \alpha\underline{y}_1$ where α is an element in the field F (usually $GF(2)$).

(3) Causal. The existence of a nonzero output, y, prior to the input of a nonzero sequence, $\underline{x}$, is forbidden. This is accomplished by the encoder output being zero when its memory elements are all zero, therefore zero inputs.

(4) Finite State. The states of the encoder are finite in number due to the value of the constraint length. Thus, a binary encoder has 2^{K-1} possible states. Each state being those input elements involved in the generation of the next output (code) bit, when combined with the next input bit entering the encoder.

d. Property 4

The chosen encoder $[\underline{G}]$ should have the minimum number of memory elements to generate the code.

e. Property 5

The chosen encoder $[\underline{G}]$ should generate a code yielding the fewest number of decoding errors per error

event (channel noise). Thereby, the fastest <u>correct</u> decoding decision is made at the receiver.

The preceding paragraphs provide a basis by which the acceptable class of encoders is greatly reduced in number. However, the selection of $[\underline{G}]$ in its optimum form for a given channel is a long drawn out process, possibly best suited for computer analysis. Theoretical selections have been made by many people, [Ref.], who based their decisions on the free distance alone.

3. <u>Representation</u>

The encoder configurations used to this point are accurate physical descriptions, but they lack a convenient form needed for analysis. There are five encoder representations which show varying degrees of the code's structure. Each is discussed in the following sections.

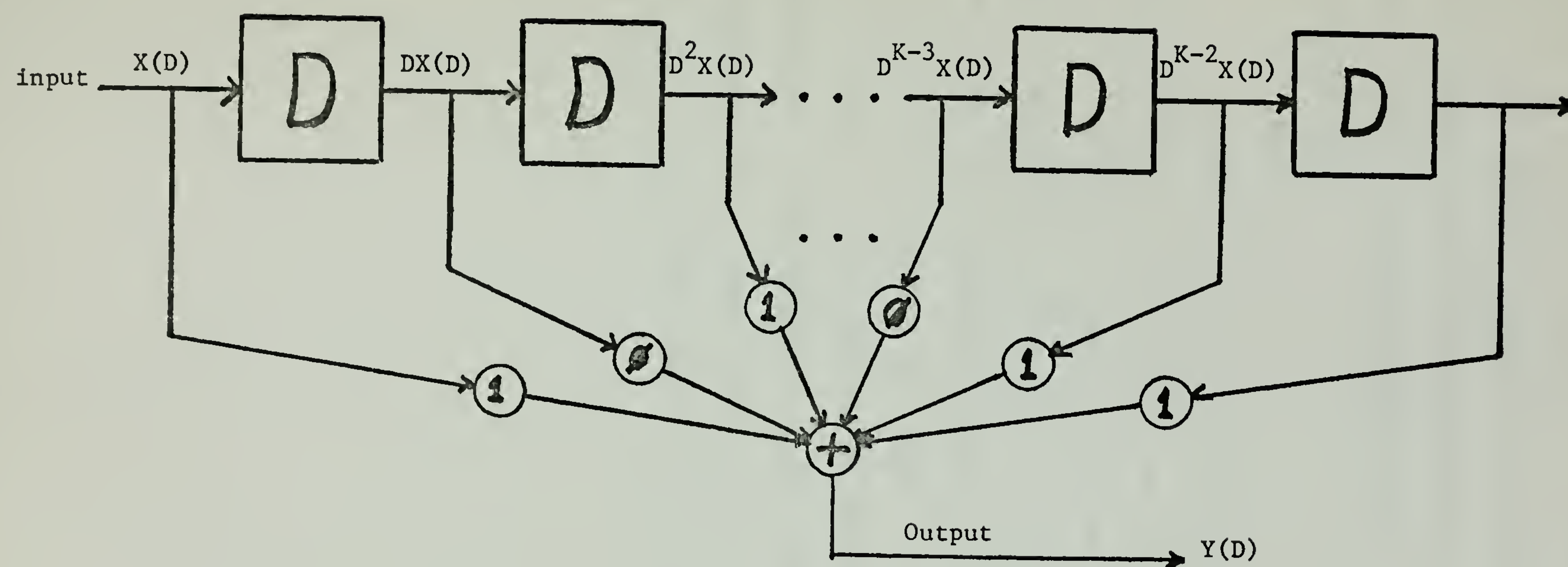
a. Polynomial

Using the D-transform, the input and output sequences appear as polynomials of degree much greater then the constraint length (K).

$$\text{Input,} \quad \underline{X}(D) = \ldots + x_{-1}D^{-1} + x_0 + x_1 D + x_2 D^2 + \ldots .$$

$$\text{Output,} \quad \underline{Y}_i(D) = \ldots + y_{i,-1}D^{-1} + y_{i,0} + y_{i,1}D + y_{i,2}D^2 + \ldots ,$$

where $1 \leq i \leq n$. The generator sequence, for the Mealy machine configuration in figure 7, is a polynomial of degree equal to K-1. The mapping of $\underline{X} \to \underline{Y}_i$ is accomplished by polynomial multiplication.



Where $Y(D) = (1+D^2 + \ldots + D^{K-2} + D^{K-1})X(D)$

Figure 7. Polynomial Representation of a Generalized Generator Sequence.

$$\underline{Y}_i(D) = \sum_{j=0}^{L} (x_j \cdot g_{i,p-j}) D^p ,$$

where L is the bit length of the input sequence. This representation has little advantage in a code structure analysis, but it is very definitive as to the procedure for code generation.

b. Matrix

A matrix, $[\underline{G}]$ representation of a rate 1/n encoder starting at state zero and time zero is achieved by the manipulation of each generator sequence. When $\underline{y} = [\underline{G}]\,\underline{x}$, $[\underline{G}]$ may be shown as a matrix whose row vectors are the generator sequences (figure 8a). Using this notation restricts $\underline{x}$ to be a length K. However, the requirement to change $\underline{x}$ for each code computation is eliminated by the matrix in figure 8b. This notation incorporates the shifting of $\underline{x}$ past each $\underline{g}_i (1 \leq i \leq n)$ into the matrix $[\underline{G}]$. For example, (figure 8 follows the example),

$$\underline{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

$n = 2$

$K = 3$

$\underline{g}_1 = (1\ 0\ 1)$

$\underline{g}_2 = (1\ 1\ 1)$

$$[\underline{G}] = \begin{bmatrix} 11 & 01 & 11 & 00 & 00 & \cdots \\ 00 & 11 & 01 & 11 & 00 & \cdots \\ 00 & 00 & 11 & 01 & 11 & \cdots \\ 00 & 00 & 00 & 11 & 01 & \cdots \\ \cdot & \cdot & \cdot & 00 & 11 & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot & \cdot \end{bmatrix}$$

$$\underline{y} = [\underline{G}]\,\underline{x} = \begin{bmatrix} 11 \\ 01 \\ 11 \\ 11 \\ 01 \\ 00 \\ 10 \\ 01 \\ 10 \\ 00 \\ 10 \\ 01 \\ 10 \\ 11 \\ 11 \\ 01 \\ 11 \\ 00 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \text{ where } \underline{y}_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \quad \underline{y}_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

This form allows for a fast generation of the code and is definitely a model which could be implemented on a computer with little difficulty. Again, there is a lack of insight into the code structure except for the possible application of matrix algebra theory.

c. Tree

The most common representation is a tree diagram (figure 9) which incorporates the branch and nodal properties of the code generation. The base branch corresponds to the initial state of the encoder prior to a non-zero entry. The first node (a) is state zero for the encoder. When a 1 begins the sequence the first lower branch is chosen and the code is found on this branch leading to node (b) or state 1. The subsequent input bits dictate whether the tree is followed up or down,

$$
\begin{array}{ccccc}
\text{Encoder Matrix} & & \text{Encoder State} & & \text{Encoded Output} \\
\begin{bmatrix} g_0^{(1)} & g_1^{(1)} & g_2^{(1)} \\ g_0^{(2)} & g_1^{(2)} & g_2^{(2)} \end{bmatrix} & \cdot & \begin{bmatrix} x_o \\ x_1 \\ x_2 \end{bmatrix} & = & \begin{bmatrix} y^{(1)} \\ y^{(2)} \end{bmatrix}
\end{array}
$$

(a)

$$
\begin{bmatrix}
g_0^{(1)} & g_0^{(2)} & g_1^{(1)} & g_1^{(2)} & g_2^{(1)} & g_2^{(2)} & 0 & 0 & \cdots \\
0 & 0 & g_0^{(1)} & g_0^{(2)} & g_1^{(1)} & g_1^{(2)} & g_2^{(1)} & g_2^{(2)} & \cdots \\
0 & 0 & 0 & 0 & g_0^{(1)} & g_0^{(2)} & g_1^{(1)} & g_1^{(2)} & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & g_0^{(1)} & g_0^{(2)} & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{bmatrix}
\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}
\begin{bmatrix}
y_0^{(1)} & y_0^{(2)} \\
y_1^{(1)} & y_1^{(2)} \\
y_2^{(1)} & y_2^{(2)} \\
y_3^{(1)} & y_3^{(2)} \\
y_4^{(1)} & y_4^{(2)} \\
\cdot & \cdot \\
\cdot & \cdot \\
\cdot & \cdot
\end{bmatrix}
$$

(b)

Figure 8. Generalized Matrix Representation of a Rate 1/2 Convolutional Encoder (K=3).

corresponding to inputs of 0 and 1 respectively. If this is continued to the K^{th} branch, all of the possible input sequences of K bits have been represented, thus each possible encoder state. Beyond the K^{th} branch it is readily recognized that the code symbols on the branches leaving from the two nodes labeled (a) are identical identical to those at the base of the tree. Even though the structure has become recurrent, for long sequences this diagram may become too large to analyze easily.

d. Trellis

Since the input sequence (1 0 0 ...) and (0 0 0 ...) generate the same n-bit codewords after the third branch (figure 9), then both nodes labeled a can be joined together. The recursive character of the tree diagram lends itself to be redrawn with remerging branches, thus forming a trellis (criss-cross) diagram (figure 10). The encoder state is the basis for the trellis diagram. These states represent those input bits in the memory cells which will generate the code bits when the next input bit enters the encoder. Therefore, there are 2^{K-1} states on either side of the diagram representing the present and next states. The branches connecting the states denote the code bits being generated when a 1 (dashed line) or a 0 (solid line) enters the encoder. Again, after the third input bit the complete trellis diagram is specified and is repeated for all succeeding branches.

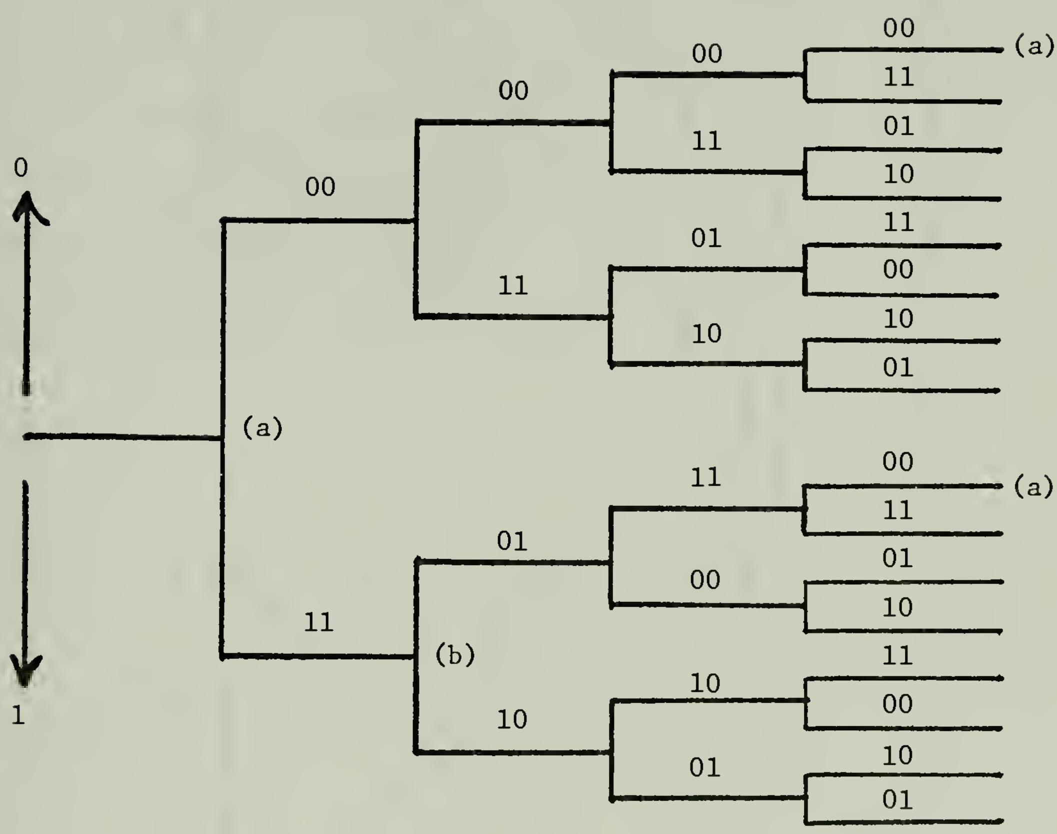
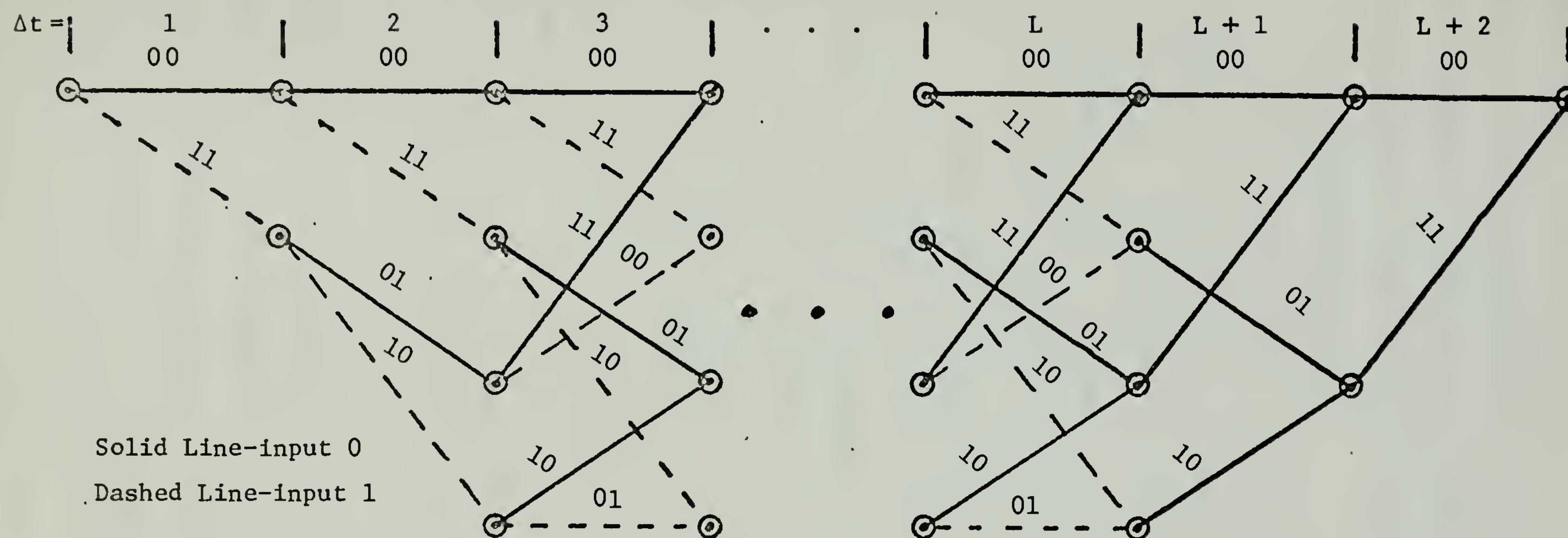


Figure 9. Tree Representation of Convolutional Encoder in Figure 5.



L = Length of input sequence
at Δt = 3, L; shows repetitive trellis structure

Figure 10. Trellis Representation of Convolutional Encoder in Figure 5.

For short constraint lengths $(K < 10)$, this representation is highly desirable due to its compactness. The trellis diagram is used in the Viterbi decoding algorithm and easily lends itself to be listed in a computer program.

e. State Diagram

The last encoder representation is similar to the trellis, but is spread out more to allow more parameters to be placed on the branches between states (nodes). The state diagram (figure 11) has the input 0 and 1 denoted in the same manner as for the trellis, but some additional notation is placed on the branches.

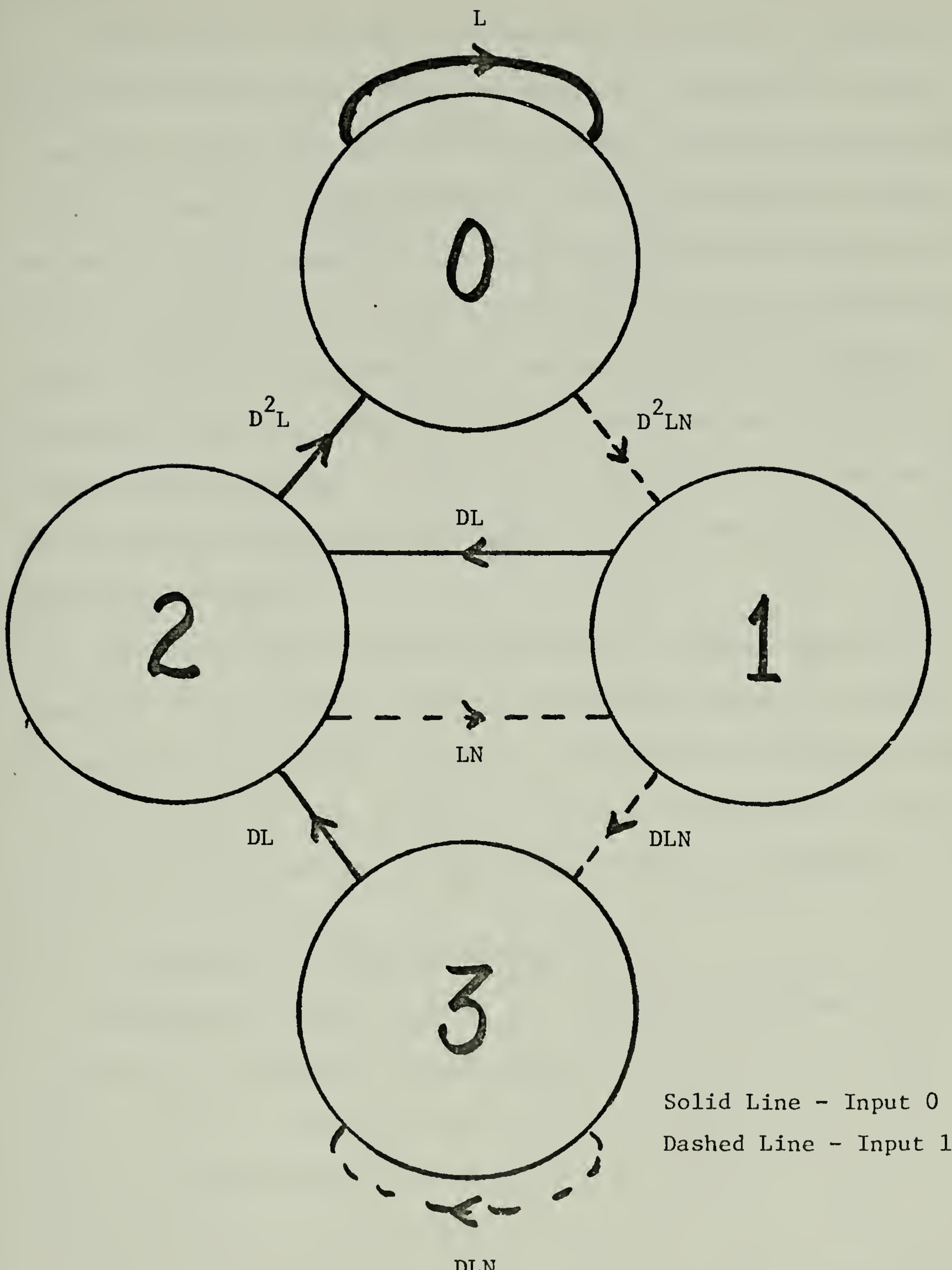
(1) $\underline{D^q}$. The value of q denotes the weight of the code bits for that branch.

(2) $\underline{L^r}$. The value of r denotes the branch length of a path from state x to state y.

(3) $\underline{N^s}$. The value of s denotes the number of input one branches encountered in a path from state x to state y.

All of the above notation is used in figure 11 and the results of a state path for the input sequence (1 0 1 0 0) are shown.

Since the state diagram is a directed graph, a transfer function can be determined using the theory of directed graphs, [Ref. 4, pp 2A-1-2A-11]. The transfer function consists of various powers of the three measurements listed above. These values are used to determine the properties (D, L, N) for all paths in the convolutional code, when the transfer function is in expanded form.



$\underline{X}$ = (10100); Path Parameters = $D^6L^5N^2$

Figure 11. State Diagram Representation of Convolutional Encoder in Figure 5.

The trellis and state diagrams are the most complete representations and offer the analyst a chance to study the path structure of the convolutional code. Each diagram yield a compact representation, easily arranged for small values of K. However, the looping branches necessary for the construction of a state diagram become very confusing and difficult to arrange for larger constraint lengths. Therefore, the trellis diagram is not only easy to construct and understand, but if the values of D, L, N, are needed for analysis they can incorporated into the diagram in an orderly fashion.

The discussion of convolutional encoding principles was basic and is by no means a complete detailed course in convolutional codes. However, the purpose of this section is to introduce the reader to concepts which will aid him in his understanding of the thesis computer program.

B. PRINCIPLES OF VITERBI DECODING

Decoding is the inverse operation to encoding and is intended to recover the source bits with all, or almost all, of the channel errors removed. The decoder, which may be implemented as hardware or software, utilizes the encoded bits to detect and/or correct errors. Error detection is similar in complexity to the encoding operation. Error correction, however, is inevitably a more complicated process than encoding, since the goal is to reduce the probability of a decoded bit error.

A scheme for decoding convolutional codes was proposed by Viterbi in 1967, [Ref. 8]. It was shown by Viterbi that an optimum decoding procedure existed for a statistically independent (bit to bit) input sequence transmitted over a channel whose errors occur independently from channel bit to channel bit. Subsequently, Forney found that the Viterbi decoding algorithm is synonomous with maximum likelihood sequence decoding, [Ref. 3].

1. Description

Simply stated, the Viterbi algorithm is a solution to the problem of finding the most likely encoded sequence through a state (finite) diagram representing the encoder. For a decoder to minimize the overall error probability of a decoded bit by brute force, maximum likelihood decoding would mean calculating the likelihood of the received sequence on all paths of the encoder state diagram. However, there are two factors which reduce the complexity of this problem. The first is the fixed periodic structure of the encoder trellis diagram, and the second is the code characteristic of remerging paths after the same K input bits are applied to two different paths (figure 12). For these reasons, the trellis diagram is an ideal tabular representation of the flow of the code at any instant.

The decoding process, as mentioned above, was found optimal for a statistically independent input sequence in discrete time. Along with this stipulation, the channel noise is also memoryless, as is the case for a binary

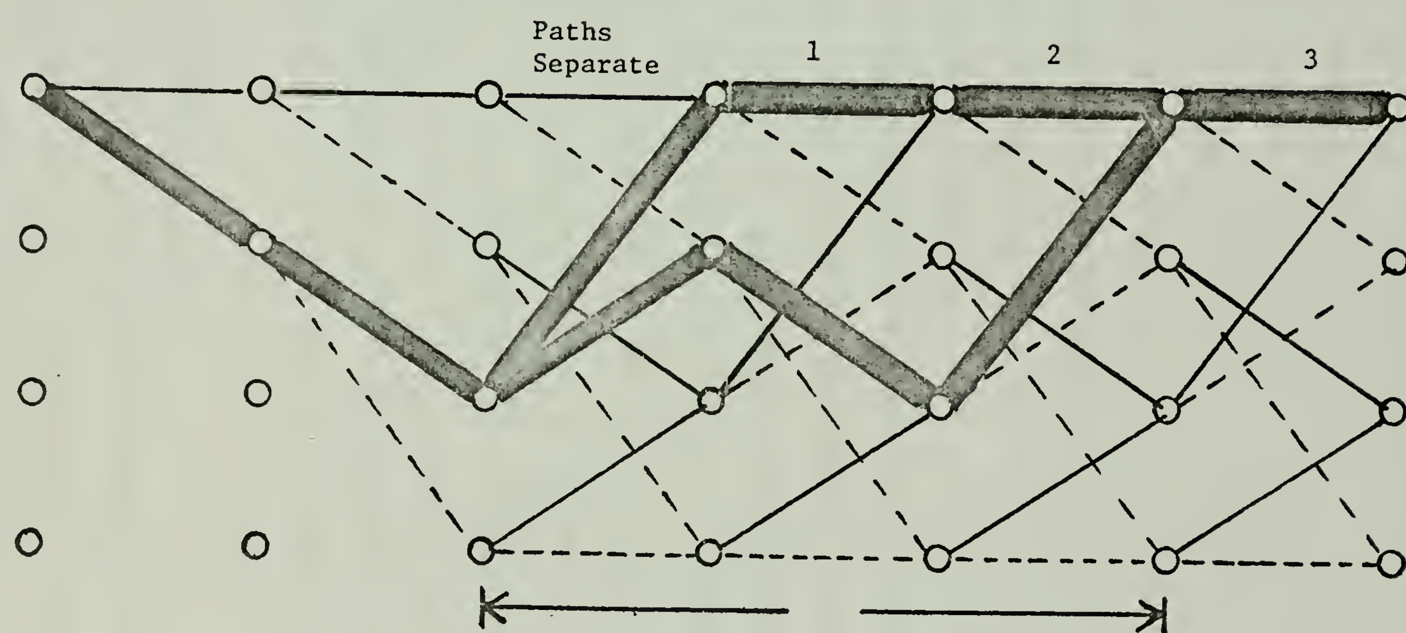


Figure 12. An Example of Remerging Path in Trellis Diagram of Convolutional Encoder (K=3).

symmetric channel (BSC) with hard decision demodulation (2^1 levels) or an additive white gaussian noise (AWGN) channel with soft decision demodulation (2^Q levels, $Q > 1$). The BSC errors transform a 0 to a 1 and a 1 to a 0 and occur independently from bit to bit with probability p. In the AWGN channel, the probability of a given quantized value (0 to 2^Q-1) of a received bit is determined from the gaussian probability density function. These values of the received bits are used to determine the most likely received sequence.

A scoring procedure is employed indicating the trellis path which shows the least difference from the received sequence. This is accomplished by using the concept of distance for the BSC and a numerical difference for the AWGN channel. The latter may be called the innerproduct of the received bit and the calculated (trellis) transmitted bit. This is implemented digitally by taking the exclusive OR of the two Q bit representation of the bits above. Both (BSC and AWGN) techniques yield a value that can be used to determine the most likely (lowest score) path.

2. Implementation

In order to describe effectively the implementation of the Viterbi decoding algorithm an example follows with step by step explanations supplemented by appropriate diagrams. This example will be for a rate 1/2 convolutional code with K = 3. The generator sequences are 5_8(1 0 1)

and 7_8 (1 1 1). The following sequence is the encoder input: (1 0 1 1 0 0 0 1 1 0 1 0 0 0 1 0 ...).

The corresponding code sequence is (11 01 00 10 10 11 00 11 10 10 00 01 11 00 11 01 11 00 ...). An error sequence denoting BSC noise perturbs this sequence prior to being received at the decoder. The error sequence is (01 00 00 00 01 00 00 00 00 10 00 00 00 00 00 10 00 00 ...). Then, the received sequence is (10 01 00 10 11 11 00 11 10 00 00 01 11 00 11 11 11 00 ...).

The following subsections are the steps of the Viterbi algorithm decoding the perturbed (received) sequence.

a. Step 1

The following figure and tables are set up to indicate the trellis structure, the path (survivor) sequences and the path scores.

(1) Trellis. (Figure 10).

(2) SSEQ(I). The present state survivor sequence.

(3) SCORE(I). The present state survivor sequence score.

(4) SSEQ(J). The next state survivor sequence.

(5) SCORE(J). The next state survivor sequence score.

b. Step 2

An initialization of tables (2) and (3) is shown in figure 13. Tables (4) and (5) are not considered until the first group of 2(n) received bits enter the

State	SSEQ(I)						Score (I)
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	6
2	0	0	0	0	0	0	6
3	0	0	0	0	0	0	6

Decoder Constraint Length (DCL) = 6

Figure 13. Present State Initialization for Viterbi Decoding Algorithm.

decoder. The all zero sequences placed in SSEQ(I) depicts the assumption that the encoder had no set bits entered prior to the beginning of the message under consideration. A decoder parameter is the length of the survivor sequences used to hold path estimates until a bit is decoded with the desired probability of error. This length is called the decoder constraint length (DCL), whose value is 6(2 x K) for this example. Also initialized is SCORE(I) so that a score of zero is in state 0 to denote that the encoder is assumed to have started from this state. The other states are assigned scores which demonstrate the unlikelihood of the encoder starting from these. Normally, a value of 2 x K is sufficient, which is 6 in this example.

c. Step 3 (Figure 14)

This step is the first in the actual decoding process. However, it will become obvious that it is recursive throughout the remainder of the algorithm. Now, the first 2 received bits (10) enter the decoder. These bits are compared with each pair of code bits on the branches of the trellis and a distance (Δ) is determined for each branch. This Δ is then added to the present score of the corresponding present state. Now there are two scores coming in on the branches to each next state. At this time, a decision is made as to which branch, coming into each next state, has the lowest score. If the scores are identical then an arbitrary choice can be made, decode estimate is zero for this example or an alternative may be exercised to

State	SSEQ(J)						Score (J)
0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1
2	1	0	0	0	0	0	6
3	0	0	0	0	0	0	6
Decoder Input	10						

Present State Decoded Bit = 0 From State 0

Figure 14. Next State Tables at the Completion of the First Shift of 2-Bits Into the Decoder.

further evaluate the Δ added to the score in SCORE(I). When all the next states have a score, they are entered in SCORE(J). Then the next state's survivor sequences are determined for entry into SSEQ(J). This is accomplished by determining what bit, 0 or 1 was dropped from the present state at the end of the minimum score branch for each next state. Then, this bit is shifted into the beginning of the appropriate present state sequence (SSEQ(I)) and the entire sequence is then placed in the SSEQ(J) table, corresponding to its next state.

The oldest bit just shifted out of the sequence in SSEQ(I) at state 0 is the most likely decoded bit since it is associated with the minimum score (0) sequence in the present state. The bit decoding, after a length of DCL, minimizes the effect of randomly spread errors in the received sequence. Again, if the scores of two or more sequences are equal, then a decision is made according to a more detailed comparison of the scores in SCORE(I).

d. Subsequent Steps

Since the most recent paths and scores are now in SSEQ(J) and SCORE(J), the roles of present and next are swapped, (I)$\leftrightarrow$(J), and step 3 is carried out again. Thus, after each decoding bit decision, the (I) and (J) are swapped, and step 3 is repeated time and again. Figure 15 shows the decoding steps for four more shifts of the decoder input. The tables, after the eighteenth shift, are shown in figure 16.

2nd

State	SSEQ(J)						Score(J)
0	0	0	0	0	0	0	2
1	0	0	0	0	0	0	2
2	0	0	0	0	0	0	1
3	0	0	0	0	0	0	3
Decoder Input	01	10					

3rd

State	SSEQ(J)						Score(J)
0	0	0	0	0	0	0	2
1	1	0	0	0	0	0	1
2	0	0	0	0	0	0	3
3	0	0	0	0	0	0	3
Decoder Input	00	01	10				

4th

State	SSEQ(J)						Score(J)
0	0	0	0	0	0	0	3
1	0	0	0	0	0	0	3
2	0	1	0	0	0	0	3
3	0	1	0	0	0	0	1
Decoder Input	10	00	01	10			

5th

State	SSEQ(J)						Score(J)
0	1	0	1	0	0	0	3
1	0	0	0	0	0	0	3
2	1	0	1	0	0	0	2
3	1	0	1	0	0	0	2
Decoder Input	11	10	00	01	10		

Note: The present state tables for each shift are derived from the previous next state tables.

Figure 15. The Next State Tables for the Second Through Fifth 2-Bit Shift into the Decoder.

State	SSEQ(J)						Score (J)
0	0	1	0	0	0	$\boxed{1}$	4
1	1	1	0	0	0	1	5
2	1	1	0	0	0	1	6
3	1	1	0	0	0	1	6
	00	11	11	11	00	11	

Decoded Bit-Bold Outline

Decoded Output = (1 0 1 1 0 0 0 1 1 0)

Figure 16. The Next State Table after 18 Shifts and the Decoded Output to that Point.

It is noticed, at this time, that the first 8 (DCL + K-1) bits are zero. This is explained by the presense of 6(DCL) zeroes in the initial SSEQ(I) and the 2(K-1) zeroes in the encoder when the first nonzero code bits were generated.

Another note should be made of the ease with which a soft decision demodulation scheme could be incorporated into the scoring process. This dimension would produce a more accurate representation of the likelihood of a sequence and possibly eliminate the need for arbitrary decisions, which may also be time-consuming.

At a first glance this procedure may seem strange or awkward, but there can be no doubt that the Viterbi algorithm is easily computer (hardware or software) implementable. The recursive nature of the primary decoding step is the prime factor in controlling the size of that implementation.

III. COMPUTER PROGRAM

The term simulation in the title of this thesis should not be construed to mean the encoder and decoder operations (Programs) are simulated. By no means is this the case, for the rate $1/n$ convolutional encoder and the Viterbi decoder are software implementations that could be used in an actual system. Of course, some form of synchronization is needed for practical operation, but in this program that is assumed to have been accomplished by elements (hardware or software) preceding the decoder in the receiver.

A. CHANNEL NOISE

Actually, the simulation occurs when a channel is described in the program by a noise generating section. This section includes a quantization segment, random number segment, noise generation segment, and perturbation (summing) segment. The following paragraphs are discussions of these four divisions of the channel noise program.

1. Quantization

This part of the program uses the encoded sequence as an input, then passes this sequence on to the noise sections after Q zeroes or ones have been substituted for each 0 or 1. When $Q=1$ is a program input this segment is bypassed for apparent reasons.

2. Random Numbers

The random numbers generated in this program were obtained by using the Lehmer congruential method.

$$X_{n+1} = aX_n + b \pmod{T_0},$$

where $a = 257$, $b = 1$, and $T_0 = 2^{16}$. The term X_0, starting number, is varied to provide representative sequences of the distribution shown in figure 18. These numbers are used to determine the time between set bits in noise sequence. The period of the random number sequence is 2^{T_0}, therefore, a large sampling of the sequence approaching 2^{T_0} would result in a binomial distribution. However, the test runs used to determine results in this thesis employed message inputs of 10,000 bits in length. Thus, the distribution (dots) shown in figure 17 is a better representation of the time between set bits in the noise sequence.

3. Noise Generation

After a random sequence has been generated in the previous segment, 2. , a noise sequence is formed using the numbers. Multiplication of the number of set bits in 16 bits (T), the length of one random number, with powers of $2(2^{-2},2^{-1},2^{0},2^{1})$ determine how many zeroes will occur before the next set bit in the noise sequence. For example, if the random number being considered by the program is 0110011100100100, then its weight is 7_8 or 111_2. If the noise parameters (core locations 10230 and 10236) specify multiply by 2^{-1}, then the number 7_8 is shifted one place right and becomes 3_8. Likewise, if multiply by 2^1 is entered, then 7_8 becomes 16_8. These new numbers are then used to determine the space between set bits in the noise sequence. Each 16-bit number from the congruential random

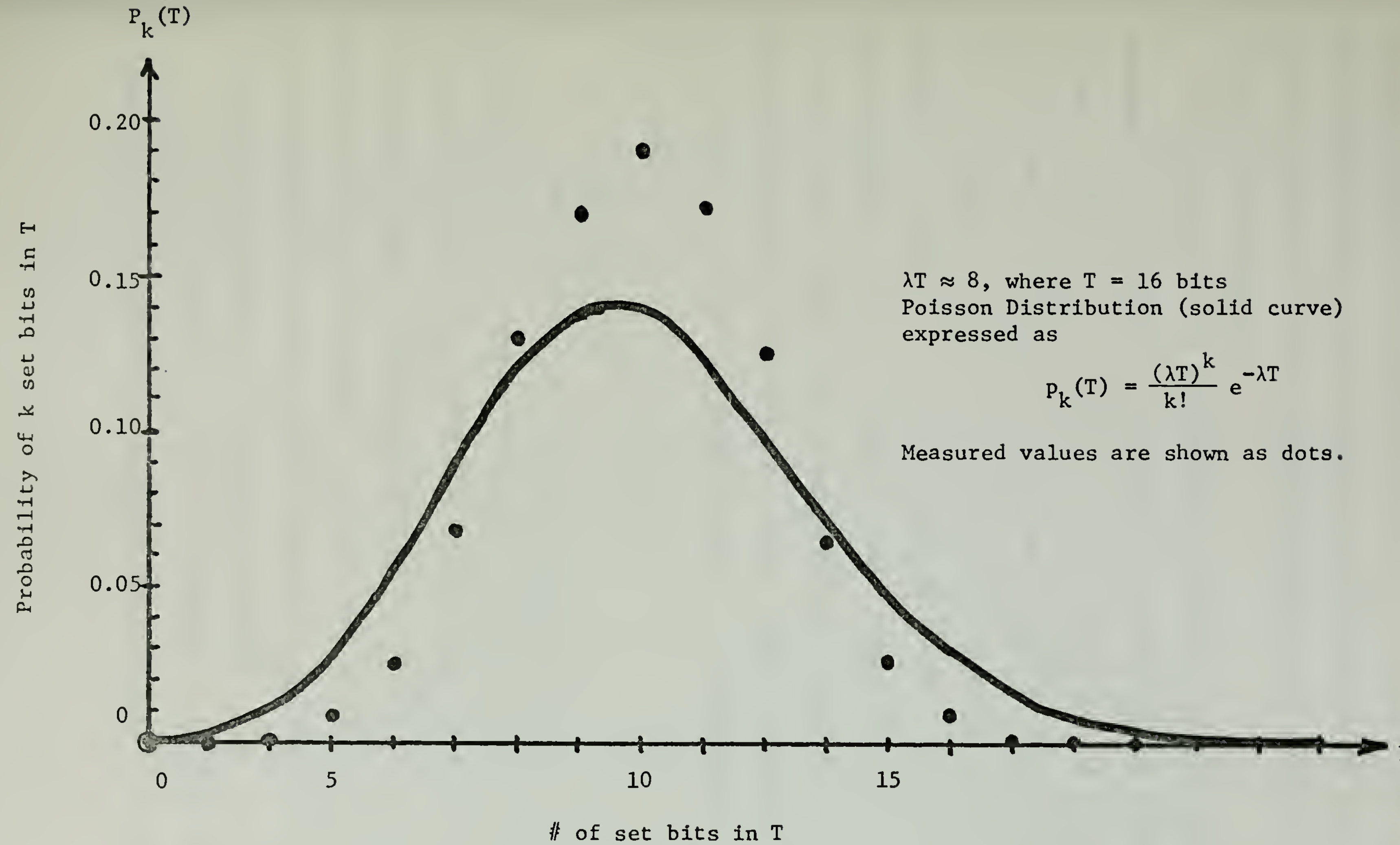


Figure 17. Comparison of Congruential Random Number Generator (10^5 bits) with Poisson Distribution.

number generator is used to generate a space. The number of 16-bit numbers used in one run is determined by the number of code bits which are present when the noise is added to the code sequence.

The variation in the set bit density for the noise sequence is analogous to varying the density of channel errors. This fact is used in the evaluation of the codes in section IV. Figures 18 through 23 show the probability of k errors in 16 channel bits ($Q = 1,2,3$) versus the number of errors, k, in 16 channel bits. The plots are for averages (λT) of about 10 and 2 errors per 16 channel bits. Also shown is the corresponding plot of the Poisson distribution as a continuous curve. Any discrepancy between the theoretical and measured (dots) is attributed to the 16 bit length constriction primarily.

4. Perturbation

The purpose of this last segment of the noise simulation is to simply exclusive OR (additive noise) the noise sequence to the quantized encoded sequence. This operation forms a perturbed sequence, which is the received sequence for the decoder. If this was to be changed to multiplication or some other operation another noise form could be simulated for the coding scheme.

All of the details, as to programming specifics, are discussed in Appendix B, along with a program listing.

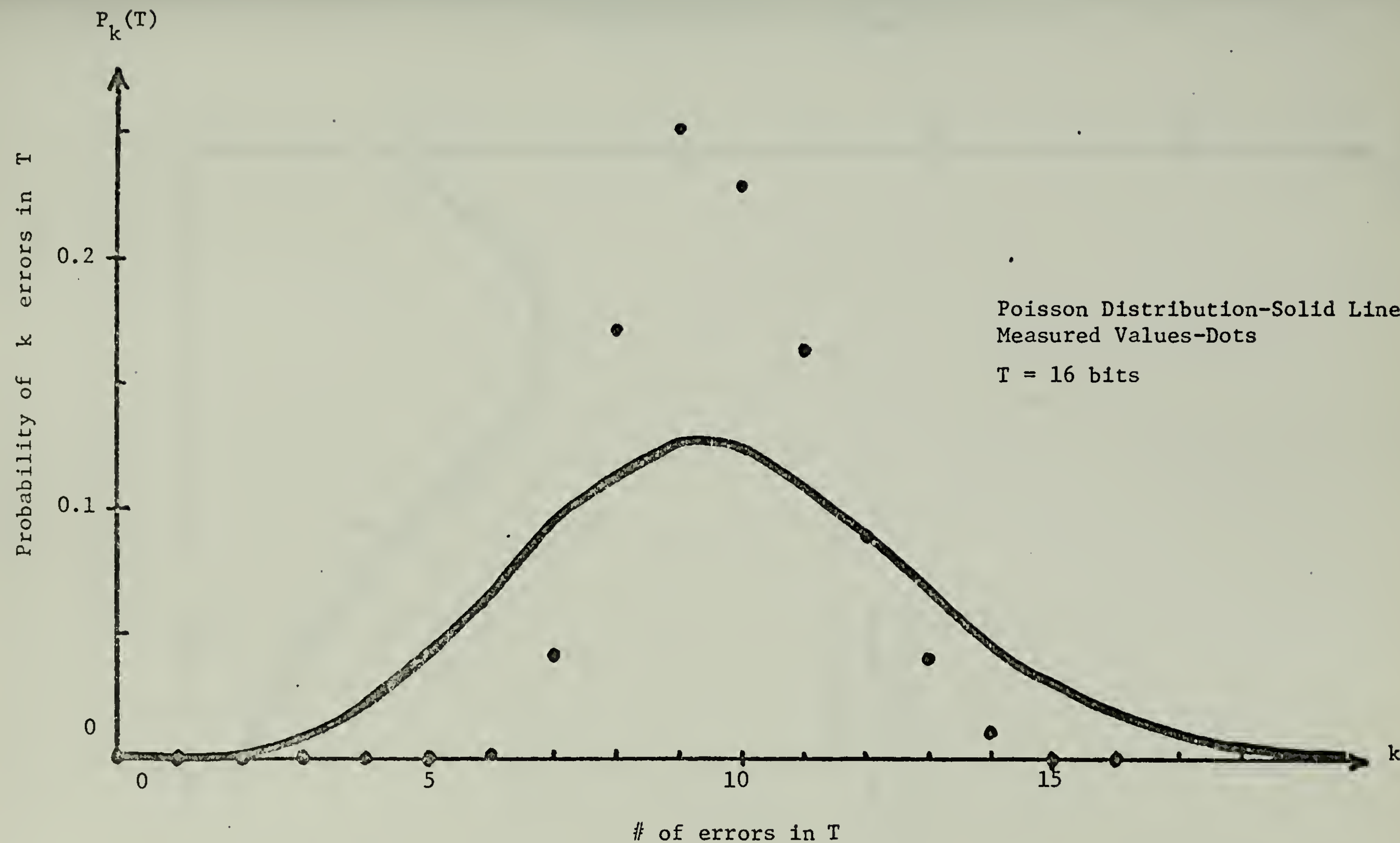


Figure 18. Simulated Noise (Q = 1, λT = 9.8) vs Poisson Distribution (λT = 9.8).

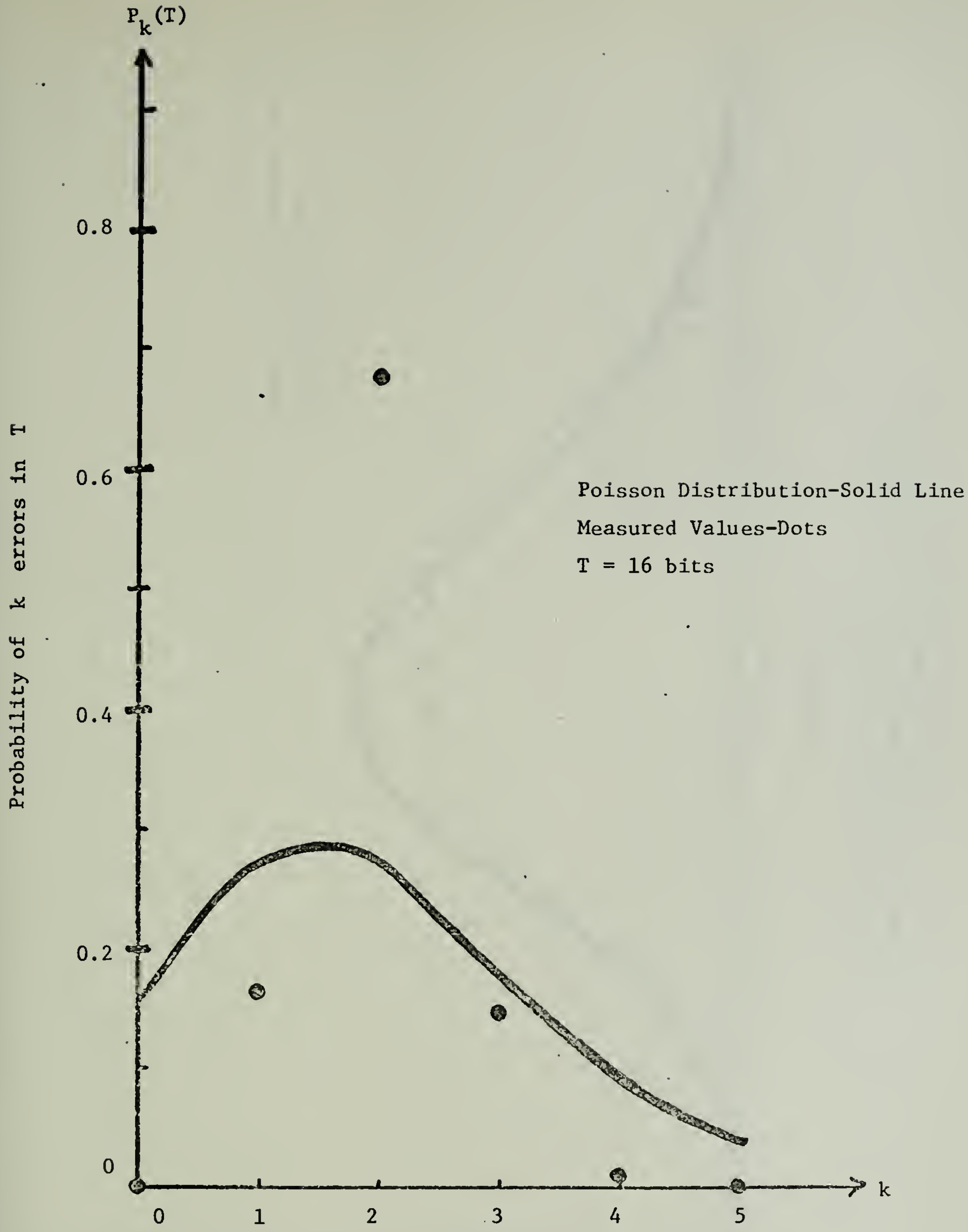


Figure 19. Simulated Noise ($Q = 1, \lambda T = 2.0$) vs Poisson Distribution ($\lambda T = 2.0$)

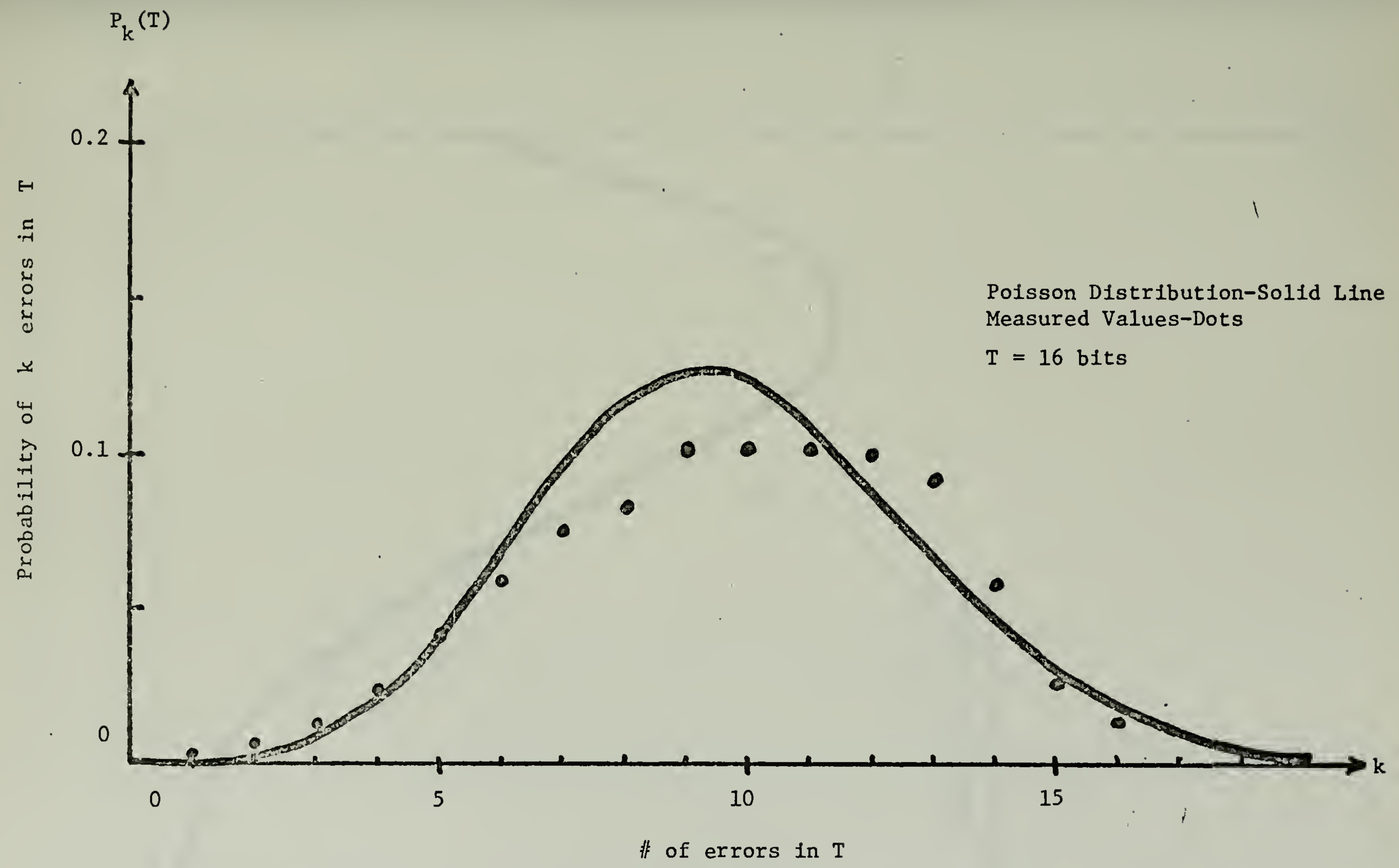


Figure 20. Simulated Noise ($Q = 2$, $\lambda T = 9.8$) vs Poisson Distribution ($\lambda T = 9.8$).

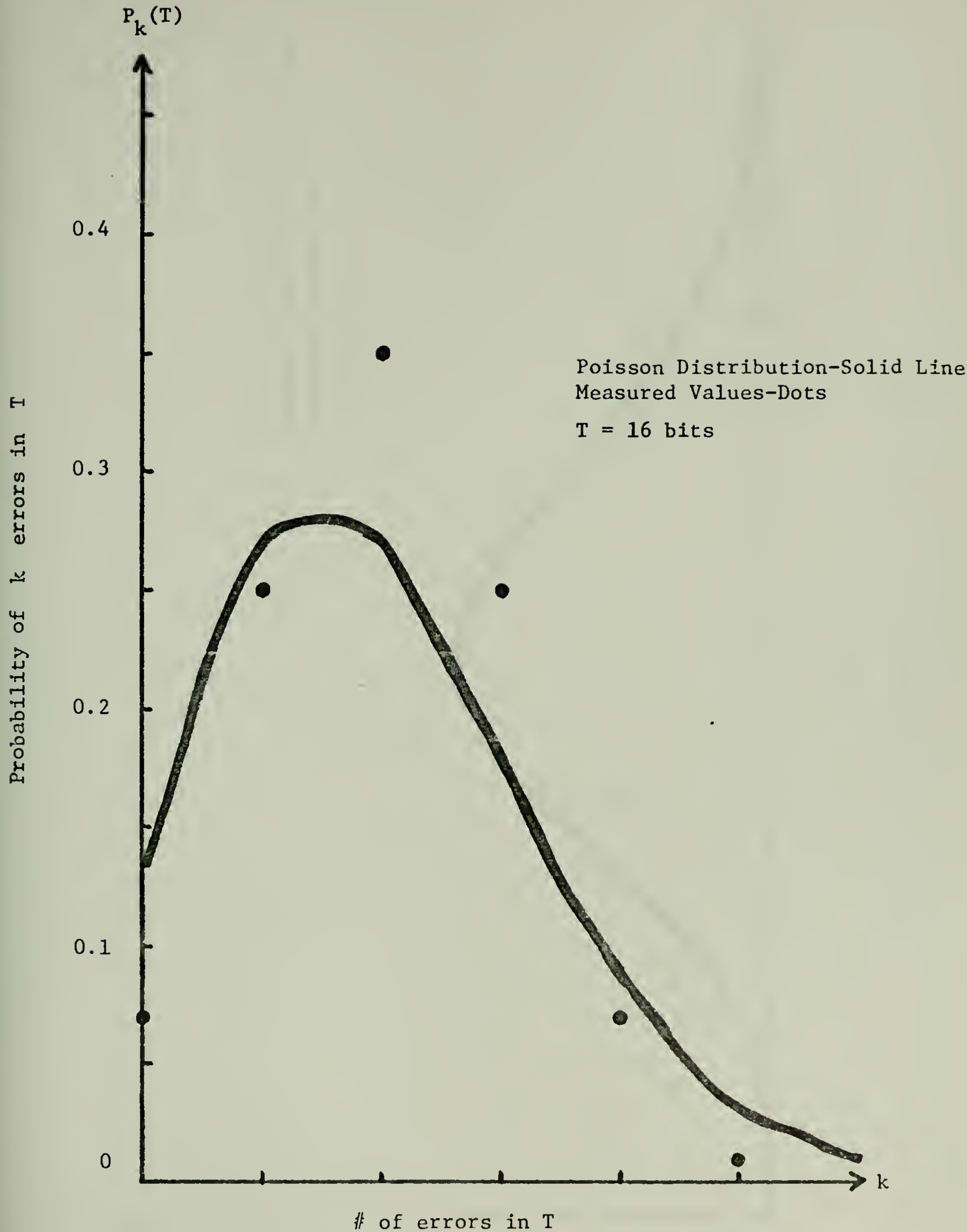


Figure 21. Simulated Noise (Q = 2, λT = 2.0) vs Poisson Distribution (λT = 2.0)

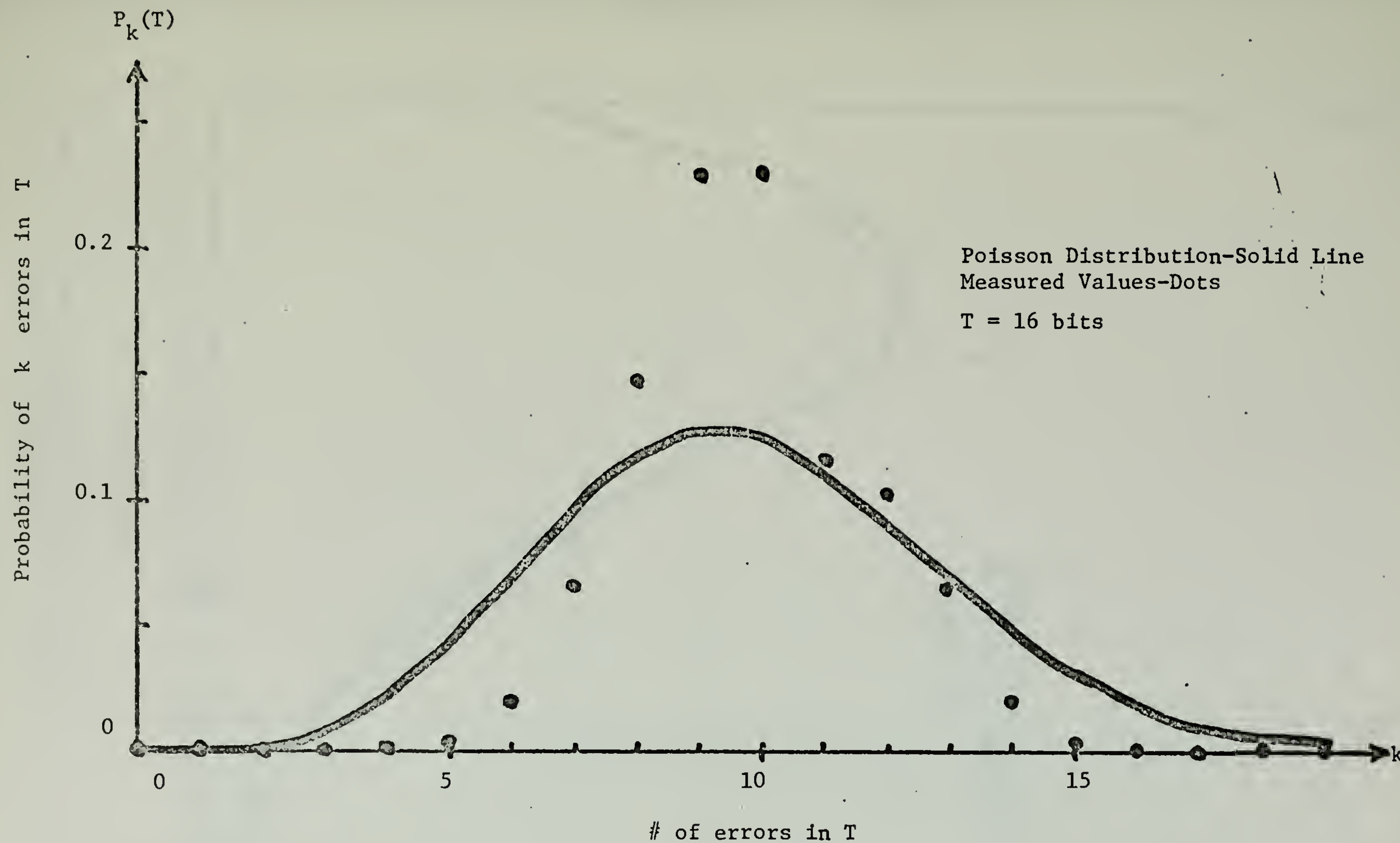


Figure 22. Simulated Noise (Q = 3, $\lambda T = 9.8$) vs Poisson Distribution ($\lambda T = 9.8$)

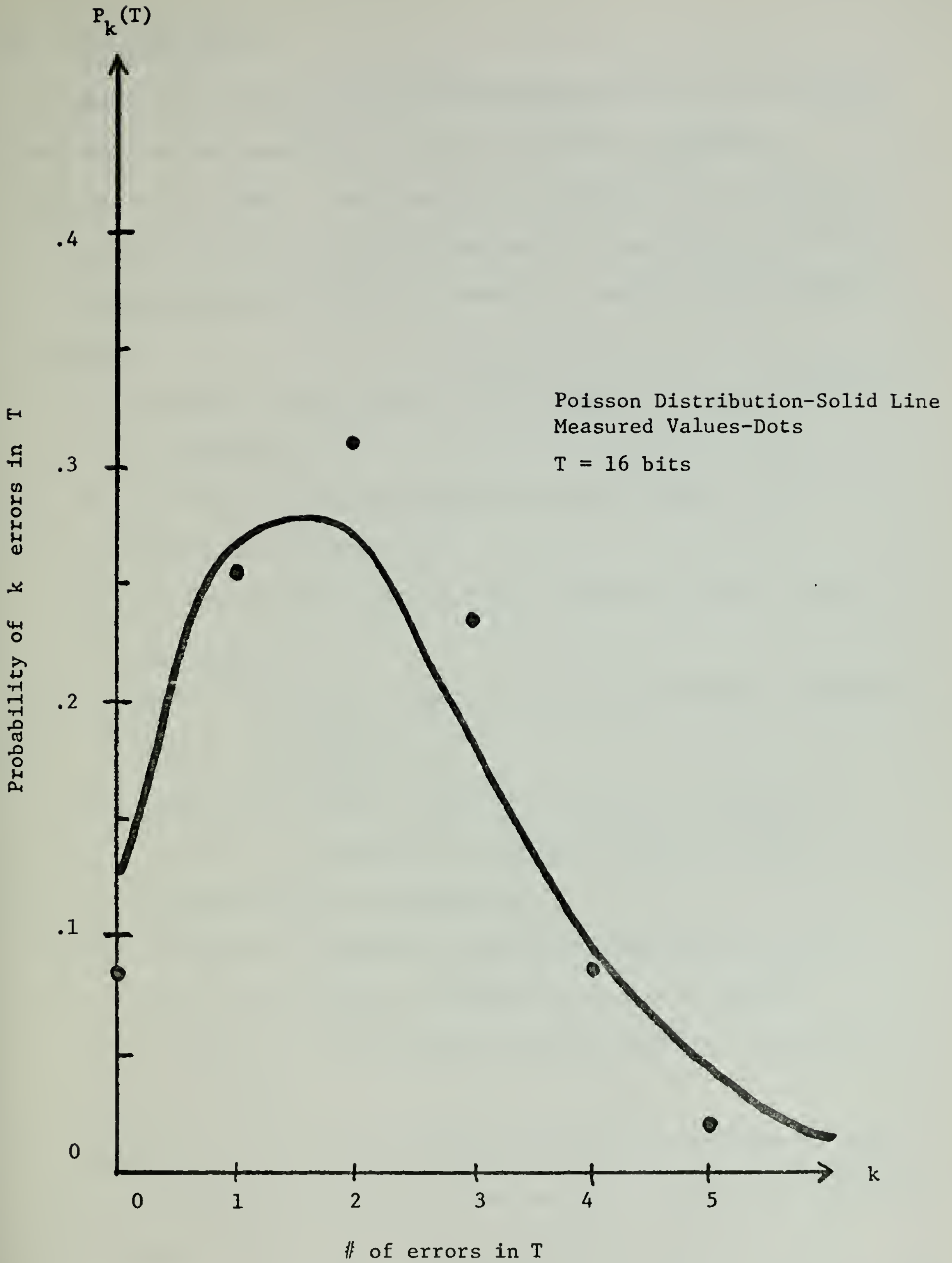


Figure 23. Simulated Noise ($Q = 3$, $\lambda T = 2.1$) vs Poisson Distribution ($\lambda T = 2.1$).

B. PROGRAM FLOW

The four major sections of the program are the encoder, decoder, noise generator, and the message generator (ASCII 7-bit code). An appendix is devoted to a detailed listing and brief discussion of each of the above sections.

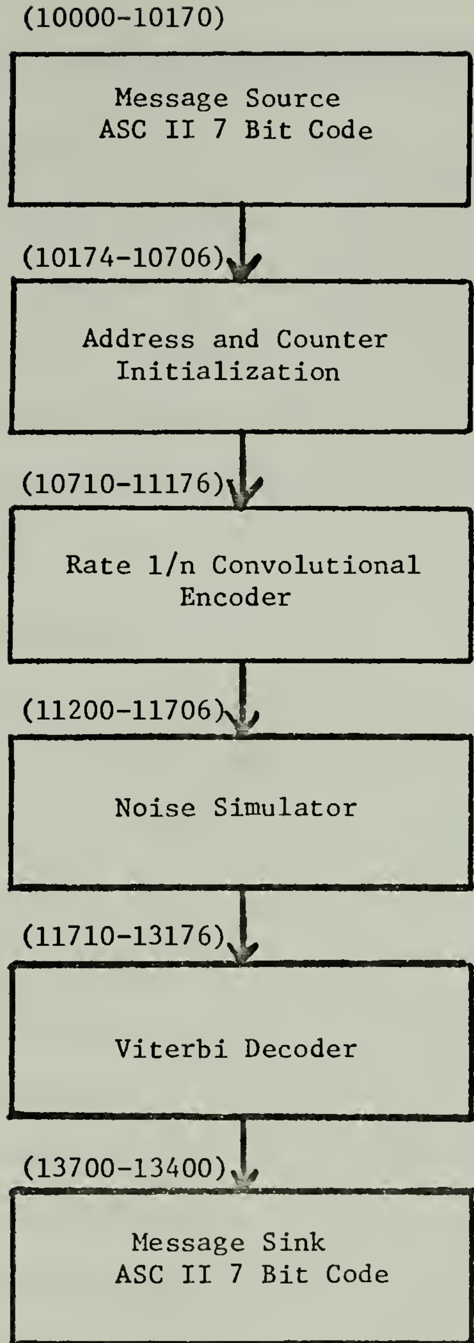
The parameters that are needed to initiate the simulation are:

1. N; the inverse rate $(1/N^{-1})$ (memory location 10200 in Appendix F).
2. K; the encoder constraint length (memory location 10206 in Appendix F0.
3. Q; the quantization levels (memory location 10214 in Appendix F).
4. DCL; the decoder constraint length (memory location 10222 in Appendix F).
5. Noise parameters; operation (÷/x) and operand (1,2,3,...)(memory locations 10236 and 10230 respectively in Appendix F).
6. Generator sequences; representations entered so that first cell in encoder is bit 0 (memory locations 13760-13776, where G_1(13760, G_2(13762,..., G_8(13776)).

Another programming concept, position independent code (PIC), is used to allow the user to move the entire program or any part (with few changes) to another part of core. This approach introduced the need for a program segment that initializes all addresses and counters (constants) used in

the program. The listing and a brief description of this segment are included in Appendix D.

The entire program, in block form, is depicted in figure 24. If the reader is interested in a detailed program description he is referred to Appendices B-E. Appendix A contains a map of that part of core used for the implementation of the position independent code.



Note: The numbers in parentheses are actual addresses occupied by the instructions corresponding to a particular flow chart block.

Figure 24. Block Diagram of Program Flow.

IV. RESULTS AND CONCLUSIONS

The term best code is used in many papers, but for the most part the use of the term best code is not defined in sufficient detail. This is one descrepancy that will be avoided by this thesis. Included in this section is a summary of the computer results obtained from a best code (defined below) determination procedure and encoder/decoder parameter variations.

A. BEST CODE DETERMINATION

The determination of a best code for a specific channel a simulated channel in this paper, follows a procedure based on Shannon's fundamental theorem for a discrete channel with noise. This procedure is best described using Shannon's representation of the attainable region in a graph (figure 25) of $H(x)$ (information rate) versus $H_y(x)$ (probability of decoded messager error). If $H(x) \le C$ (channel capacity), then Shannon shows that $H_y(x)$ can be made arbitrarily small with a proper encoding procedure. When $H(x) > C$, then the excess information being pushed onto the channel can only increase the uncertainty $(H_y(x))$ of the decoded message. The minimum value of $H_y(x)$ is very close to $H-C$ in this latter case.

The computer program in this thesis is implemented with a supplemental segment which stepped through all well-defined convolutional code generators of a specific code rate and

constraint length. Well defined refers to the constraint length definition of the code, meaning that the coefficient of the zero power in the generator polynomial must be set for at least one generator sequence, and then another sequence must have the coefficient set for the K-1 power. An example of this is shown here:

$$n = 2,\ K = 4;\quad \underline{g}_1(D) = 1 + D^2$$

$$\underline{g}_2(D) = D + D^2 + D^3$$

$$n = 3,\ K = 3;\quad \underline{g}_1(D) = D + D^3 \Rightarrow 1 + D^2$$

$$\underline{g}_2(D) = D^2 + D^3 \Rightarrow D + D^2$$

$$\underline{g}_3(D) = D + D^2 + D^3 \Rightarrow 1 + D + D^2$$

The rate is defined by the number of nonzero generator sequences.

The first two paragraphs describe a procedure and the selection of encoders to be processed by the program. The program is applied to find the generator sequences, for a given code rate/constraint length, which come the closest to the lower error $(H_y(x))$ boundary of figure 25. Code rates of 1/2 and 1/3 are tested and if the input message is assumed to have 1 bit of information in each bit entering the encoder, then these rates correspond to those plotted on the horizontal scale of figure 25. Tables II through IV present a summary of the test run results for a best code determination.

Each test run was made for a 40,000 bit encoder input applied over various samples of the simulated noise and at

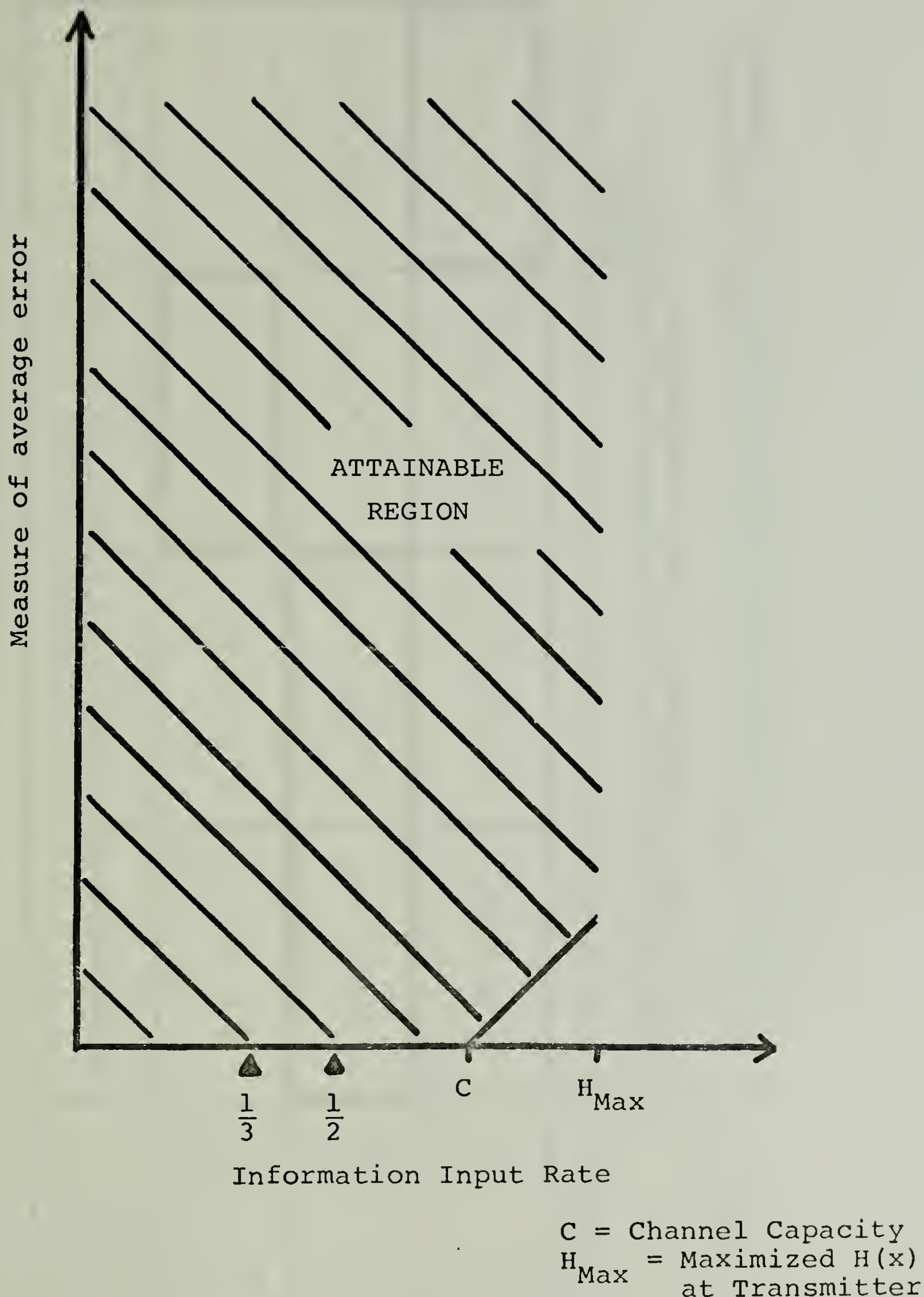


Figure 25. The measure of average error for a given information input rate to a channel (figure is similar to Shannon's Representation).

Rate $(\frac{1}{n}) = \frac{1}{2}$

Quantization (Q) = 1

Encoder Constraint Length (K)=3

Decoder Constraint Length (DCL)=16

Generator Sequences	λT(Channel errors in 16 bits)			
	1.0	2.0	4.2	9.8
$\underline{g}_1 = (100)$ $\underline{g}_2 = (001)$	5.41×10^{-2}	1.09×10^{-1}	$\boxed{2.4 \times 10^{-1}}$	5.78×10^{-1}
$\underline{g}_1 = (110)$ $\underline{g}_2 = (001)$	$\boxed{< 2.5 \times 10^{-5}}$	$\boxed{3.19 \times 10^{-2}}$	2.74×10^{-1}	5.68×10^{-1}
$\underline{g}_1 = (111)$ $\underline{g}_2 = (111)$	4.65×10^{-1}	4.67×10^{-1}	4.68×10^{-1}	$\boxed{4.37 \times 10^{-1}}$

Note: 1. The probability of decoded bit error = $\frac{\text{\# of message errors}}{40{,}000}$

2. The boxes with bold outline contain the minimum error probability determined at the corresponding λT.

Table II. Results of Test Runs to Determine the Generator Sequences (n = 2, K = 3) Yielding the Minimum Probability of Decoded Bit Error.

Rate $(\frac{1}{n}) = \frac{1}{2}$

Quantization (Q) = 1

Encoder Constraint Length (K)=4

Decoder Constraint Length (DCL)=16

Generator Sequence	λT (Channel errors in 16 bits)			
	1.0	2.0	4.2	9.8
$\underline{g}_1 = (1000)$ $\underline{g}_2 = (0001)$	5.23×10^{-2}	1.09×10^{-1}	$\boxed{2.40 \times 10^{-1}}$	5.78×10^{-1}
$\underline{g}_1 = (1110)$ $\underline{g}_2 = (0101)$	$\boxed{< 2.5 \times 10^{-5}}$	$\boxed{3.26 \times 10^{-2}}$	4.73×10^{-1}	5.51×10^{-1}
$\underline{g}_1 = (1111)$ $\underline{g}_2 = (1111)$	4.56×10^{-1}	4.59×10^{-1}	4.59×10^{-1}	$\boxed{4.58 \times 10^{-1}}$

Note: 1. The probability of decoded bit error $= \frac{\text{\# of message errors}}{40{,}000}$

2. The boxes with bold outline contain the minimum error probability determined at the corresponding λT.

Table III. Results of Test Runs to Determine the Generator Sequences (n = 2, K = 4) Yielding the Minimum Probability of Decoded Bit Error.

Rate $(\frac{1}{n}) = \frac{1}{3}$

Quantization (Q) = 1

Encoder Constraint Length (K)=3

Decoder Constraint Length (DCL)=16

Generator Sequences	λT (Channel errors in 16 bits)			
	1.0	2.0	4.2	9.8
$\underline{g}_1 = (100)$ $\underline{g}_2 = (100)$ $\underline{g}_3 = (101)$	1.13×10^{-3}	2.95×10^{-2}	$\boxed{1.35 \times 10^{-1}}$	5.89×10^{-1}
$\underline{g}_1 = (101)$ $\underline{g}_2 = (010)$ $\underline{g}_3 = (101)$	4.74×10^{-5}	1.47×10^{-2}	2.23×10^{-1}	$\boxed{4.52 \times 10^{-1}}$
$\underline{g}_1 = (111)$ $\underline{g}_2 = (001)$ $\underline{g}_3 = (001)$	$\boxed{< 2.5 \times 10^{-5}}$	$\boxed{1.65 \times 10^{-3}}$	1.6×10^{-1}	6.12×10^{-1}

Note: 1. The probability of decoded bit error = $\frac{\text{\# of message errors}}{40{,}000}$

2. The boxes with bold outline contain the minimum error probability determined at the corresponding λT.

Table IV. Results of Test Runs to Determine the Generator Sequences (n = 3, K = 3) Yielding the Minimum Probability of Decoded Bit Error.

four values of λT(1,2,4,10 errors per 16 bits). The computer output is a listing of the decoded message errors for each well defined encoder. This output would require many additional pages to list in this thesis. Therefore, the summary in Tables II through IV is presented instead. The number of decoded errors is then divided by 40,000 to give the measure of average error for each test run (λT).

The various values of λT are obtained by the noise simulation and have the characteristic distributions of those curves presented in figures 18 and 19.

A discussion of these results is presented in part C of this section.

B. DECODER PARAMETER VARIATIONS

In order to provide an insight as to the effects of quantization (Q) and decoder constraint length (DCL), tables V and VI are shown. The effect of increasing Q or DCL is to decrease the frequency of errors thus improving the error correcting scheme. The codes used in these tables are those chosen as the best codes from tables II through IV.

C. DISCUSSION AND CONCLUSIONS

The determination of a best coding scheme for a specific channel application can be made from tables II-IV. The generator sequences which have the minimum frequency of errors (in place of $H_y(x)$) for the various values of λT are outlined boldly. Using this comparison technique plus a comparison of the general statistics of the encoder at

other values of λT, the following generator sequences are chosen to be the encoders which meet the criteria set forth previously.

$$N = 2,\ K = 3;\quad \underline{g}_1 = (110),\ \underline{g}_2 = (001)$$

$$N = 2,\ K = 4;\quad \underline{g}_1 = (1110),\ \underline{g}_2 = (0101)$$

$$N = 3,\ K = 3;\quad \underline{g}_1 = (111),\ \underline{g}_2 = (001),\ \underline{g}_3 = (001)$$

These encoders were also used for the results obtained by varying Q and DCL in tables V and VI.

With actual concrete measurements of the effectiveness (minimum error) of a code and a means (computer program) of applying these codes to actual channel recordings there is no reason to have to settle for an inadequate error correcting system. The results presented in this section indicate clear preferences in choosing certain codes to accomplish desired communications in a given noise environment. The user specifics the task and with the computer a code can be chosen. The true value of such a program should be evaluated when implemented with actual channel noise. No one has been able to find a relation or algorithm for encoding that would enable a communication system to reach the maximum average information rate (C) with an arbitrarily small frequency of error. However, computer analysis offers the channel user an opportunity to find a coding scheme which meets the requirements for that channel.

Decoder Constraint Length = 16 Bits

Generator Sequences	Q	λT (noise errors in 16 bits)			
		1.0	2.0	4.2	9.8
$\underline{g}_1$ = (110)	1	$< 2.5 \times 10^{-5}$	3.19×10^{-2}	2.74×10^{-1}	5.68×10^{-1}
$\underline{g}_2$ = (001)	3	$< 2.5 \times 10^{-5}$	3.42×10^{-3}	6.72×10^{-2}	5.67×10^{-1}
$\underline{g}_1$ = (1110)	1	$< 2.5 \times 10^{-5}$	3.26×10^{-2}	4.73×10^{-1}	5.51×10^{-1}
$\underline{g}_2$ = (0101)	3	$< 2.5 \times 10^{-5}$	1.74×10^{-4}	8.67×10^{-2}	4.85×10^{-1}
$\underline{g}_1$ = (111) $\underline{g}_2$ = (001) $\underline{g}_3$ = (001)	1	$< 2.5 \times 10^{-5}$	1.65×10^{-3}	1.6×10^{-1}	6.12×10^{-1}
	3	$< 2.5 \times 10^{-5}$	$< 2.5 \times 10^{-5}$	2.12×10^{-2}	4.75×10^{-1}

Note: The probability of decoded bit error = $\frac{\text{\# of errors}}{40{,}000}$

Table V. Results of Test Runs for Variation in Receiver Quantization (Q) Using the Best Codes from Tables II, III, and IV.

Quantization = 1

Generator Sequences	DCL	λT (noise errors in 16 bits)			
		1.0	2.0	4.2	9.8
110	8	1.0×10^{-4}	3.19×10^{-2}	2.74×10^{-1}	5.68×10^{-1}
001	16	$< 2.5 \times 10^{-5}$	3.19×10^{-2}	2.74×10^{-1}	5.68×10^{-1}
1110	8	2.5×10^{-5}	4.43×10^{-2}	4.79×10^{-1}	5.50×10^{-1}
0101	16	$< 2.5 \times 10^{-5}$	3.26×10^{-2}	4.73×10^{-1}	5.51×10^{-1}
111	8	$< 2.5 \times 10^{-5}$	1.72×10^{-3}	1.62×10^{-1}	6.20×10^{-1}
001 001	16	$< 2.5 \times 10^{-5}$	1.65×10^{-3}	1.6×10^{-1}	6.12×10^{-1}

Note: The probability of decoded bit error = $\frac{\#\text{ of errors}}{40{,}000}$

Table VI. Results of Test Runs for Variation in Decoder Constraint Length (DCL) Using the Best Codes from Tables II, III, and IV.

APPENDIX A

POSITION INDEPENDENT CODE CORE MAP

THE CORE LOCATIONS OF ADDRESSES, COUNTERS, AND MISCELLANEOUS STORAGE (WORK) AREAS ARE LISTED BELOW. SOME OF THESE VALUES, PRECEDED BY *, ARE ENTERED BY OR COMPUTED BY THE PROGRAM INITIALIZATION SUBPROGRAM, WHICH FOLLOWS THE INPUT SUBPROGRAM.

```
*  13600;  INPUT BLOCK ADDRESS
*  13602;  ENCODED/DECODED BLOCK ADDRESS
*  13604;  QUANTIZED CODE SEQUENCE BLOCK ADDRESS
*  13606;  NOISE SEQUENCE BLOCK ADDRESS
*  13610;  RANDOM NUMBER/PERTURBED SEQUENCE BLOCK ADDRESS
*  13612;  TRELLIS TABLE BLOCK ADDRESS
*  13614;  SSEQ(I) TABLE BLOCK ADDRESS
*  13616;  SCORE(I) TABLE BLOCK ADDRESS
*  13620;  SSEQ(J) TABLE BLOCK ADDRESS
*  13622;  SCORE(J) TABLE BLOCK ADDRESS
   13624;  "NOT USED"
*  13626;  GENERATOR SEQUENCE BLOCK ADDRESS
*  13630;  STACK ADDRESS
   13632;  HOLD (STORAGE)
   13634;  DELTA (USED IN DECODER)
   13636;  MASK (USED TO MAKE DECODED BIT DECISION)
   13640;  WORK (STORAGE)
   13642;  MESSAGE WORD (16 BITS) COUNT WORK LOCATION
   13644;  MQ BIT COUNT WORK LOCATION
```

```
   13646; DECODED BIT COUNTER
   13650; NUMBER OF SCORES COUNTER
   13652; MINIMUM SCORES ADDRESS
   13654; MINIMUM SCORE VALUE
   13656; N-BIT LOOP COUNTER
   13660; ZERO BIT SCORE VALUE
   13662; ONE BIT SCORE VALUE
   13664; SSEQ WORK LOCATION
   13666; TOTAL OF NOISE VALUES IN NOISE SEQUENCE
   13670; TOTAL NUMBER OF ERRORS IN NOISE SEQUENCE
   13672; TOTAL NUMBER OF DECODED MESSAGE BIT ERRORS
   13674; "NOT USED"
   13676; "NOT USED"
*  13700; NUMBER OF CODE BITS PER INPUT BIT, N
*  13702; ENCODER CONSTRAINT LENGTH, K
*  13704; QUANTIZATION VALUE, Q
*  13706; DECODER CONSTRAINT LENGTH, DCL
   13710; MESSAGE WORD (16 BITS) COUNT, W
*  13712; POWER OF TWO USED IN GENERATING NOISE SEQUENCE
*  13714; OPERATION (DIVIDE/MULTIPLY) USED FOR NOISE
*  13716; Q X N
*  13720; Q X N X W
*  13722; 4 X K (NONZERO STATES INITIAL SCORE)
*  13724; 2 ** K
*  13726; (2 ** K) - 1
*  13730; 2 ** (K - 1)
*  13732; (2 ** (K - 1)) - 1
```

```
*   13734; 2 ** (K - 2)

*   13736; (2 ** Q) - 1

*   13740; DCL + K - 1

*   13742; 2 ** (DCL - 1)

*   13744; SCORE(I) - SSEQ(I) (SUBTRACT ADDRESSES)

*   13746; SSEQ(J) - SSEQ(I) (SUBTRACT ADDRESSES)

*   13750; N X W

    13752; NEXT SCORE ADDRESS

    13754; NEXT SSEQ ADDRESS

    13756; STORAGE LOCATION FOR A COUNTER VALUE OF 2

    13760; G(1), GENERATOR SEQUENCE

    13762; G(2), GENERATOR SEQUENCE

    13764; G(3), GENERATOR SEQUENCE

    13766; G(4), GENERATOR SEQUENCE

    13770; G(5), GENERATOR SEQUENCE

    13772; G(6), GENERATOR SEQUENCE

    13774; G(7), GENERATOR SEQUENCE

    13776; G(8), GENERATOR SEQUENCE
```

THE WORD COUNT IN LOCATION 13710 IS PLACED THERE BY AN INSTRUCTION AT THE END OF THE MESSAGE INPUT SUB-PROGRAM. THE VALUE OF THE GENERATOR SEQUENCES IS ENTERED BY THE PROGRAMMER BEFORE STARTING A RUN.

APPENDIX B

PROGRAM LISTING OF NOISE SIMULATION

The following flow chart depicts the flow of the instructions in the following machine language listing of the noise simulation subprogram. The numbers (base 8) on the upper left of the blocks in the flow chart correspond to those instruction addresses of the subprogram mentioned in that block.

(11200-11324)

This segment quantizes the encoded block, that is, encoded 1's become Q 1's and encoded 0's become Q 0's. The result is then placed in block specified at (13604).

↓

(11330-11374)

This segment generates the random number sequence by the Lehmer Congruential Relation. These numbers are stored in a block beginning at the address specified at (13610).

↓

(11376-11546)

This segment uses the noise parameters placed in locations (10230) and (10236) to change the value of the number of set bits in a 16 bit random number, so that the density of errors (λT) in the noise is varied. The noise sequence is stored in a block beginning at an address in (13606).

↓

(11550-11646)

This segment determines the noise value total (13666) and the number of errors (13670) for a given message, i.e., a value ≥ 4 is an error for Q=3.

↓

(11650-11700)

This segment adds the noise block to the quantized block and stores the result in a perturbed block beginning at an address in (13610)

↓

END OF SUBPROGRAM

The following computer printout is the noise program just discussed.

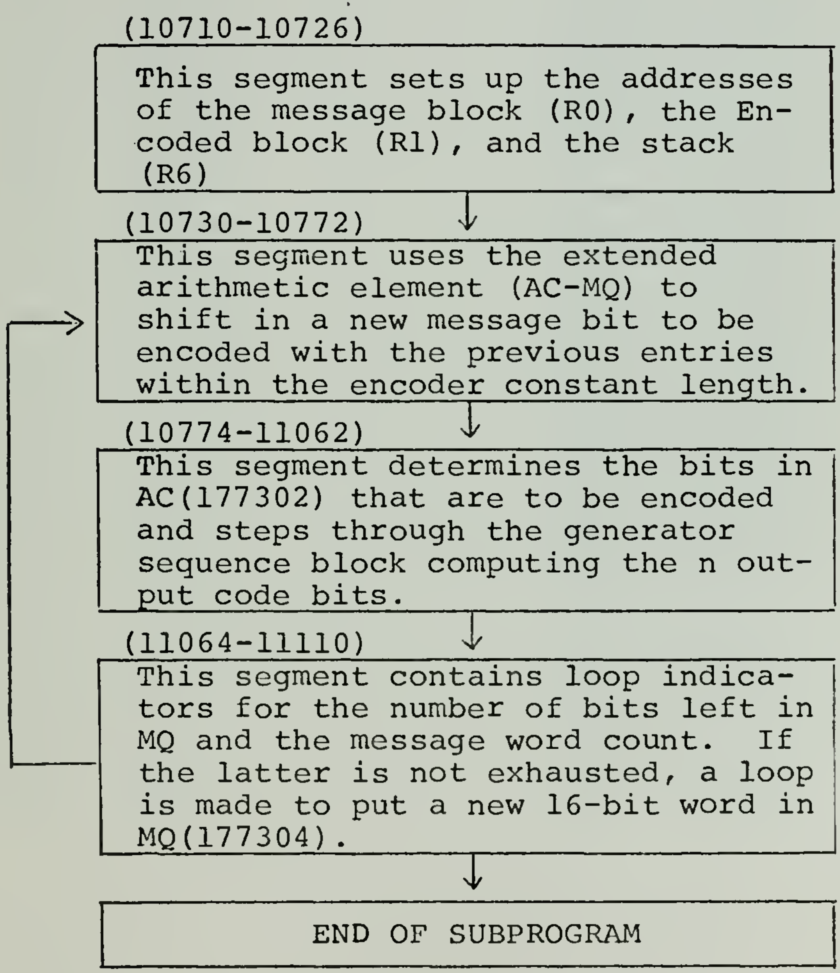
```
011200 /016700
011202 /002376
011204 /016701
011206 /002374
011210 /016702
011212 /002534
011214 /022767
011216 /000001
011220 /002462
011222 /001003
011224 /012021
011226 /077202
011230 /000436
011232 /012703
011234 /000020
011236 /012704
011240 /000020
011242 /016705
011244 /002436
011246 /006310
011250 /103405
011252 /006311
011254 /005303
011256 /001410
011260 /077504
011262 /000416
011264 /006311
011266 /005211
011270 /005303
011272 /001406
011274 /077505
011276 /000410
011300 /005721
011302 /012703
011304 /000020
011306 /000764
011310 /005721
011312 /012703
011314 /000020
011316 /000766
011320 /077430
011322 /005720
011324 /077234
011326 /000000
011330 /016700
011332 /002254
011334 /012701
011336 /004704
011340 /012702
011342 /000401
011344 /012737
011346 /044444
011350 /177304
011352 /010237
011354 /177306
011356 /005237
011360 /177304
011362 /013720
011364 /177304
011366 /077107
011370 /010067
011372 /002300
011374 /000240
011376 /005067
011400 /002264
011402 /005067
011404 /002262
011406 /016700
011410 /002176
011412 /016701
011414 /002170
011416 /012703
011420 /000020
011422 /012704
011424 /000020
011426 /012006
011430 /020067
011432 /002240
011434 /001003
011436 /016700
011440 /002146
011442 /012006
011444 /005005
011446 /006206
011450 /005505
011452 /077403
011454 /005767
011456 /002234
```

```
011460 /001005
011462 /016704
011464 /002224
011466 /006205
011470 /077402
011472 /000407
011474 /016737
011476 /002212
011500 /177304
011502 /010537
011504 /177306
011506 /013705
011510 /177304
011512 /005303
011514 /002405
011516 /006311
011520 /005305
011522 /003373
011524 /005211
011526 /000735
011530 /005721
011532 /020167
011534 /002052
011536 /001001
011540 /000403
011542 /012703
011544 /000017
011546 /000763
011550 /016700
011552 /002032
011554 /016701
011556 /002140
011560 /012702
011562 /000020
011564 /012004
011566 /016703
011570 /002112
011572 /005005
011574 /006305
011576 /006304
011600 /005505
011602 /005302
011604 /001413
011606 /077306
011610 /060567
011612 /002052
011614 /016703
011616 /002116
011620 /006203
011622 /020503
011624 /003760
011626 /005267
011630 /002036
011632 /000755
011634 /012004
011636 /012702
011640 /000020
011642 /077117
011644 /000240
011646 /000240
011650 /016700
011652 /001732
011654 /016701
011656 /001724
011660 /016702
011662 /001724
011664 /016705
011666 /002030
011670 /012003
011672 /012104
011674 /074304
011676 /010422
011700 /077505
011702 /000000
*
```

APPENDIX C

Program Listing of Rate 1/n Convolutional Encoder

The following flow chart depicts the flow of the rate 1/n convolutional encoder implementation. The corresponding machine language program follows the flow chart and the instruction addressed are indicated on top of the respective flow chart block to which they correspond in the subprogram.



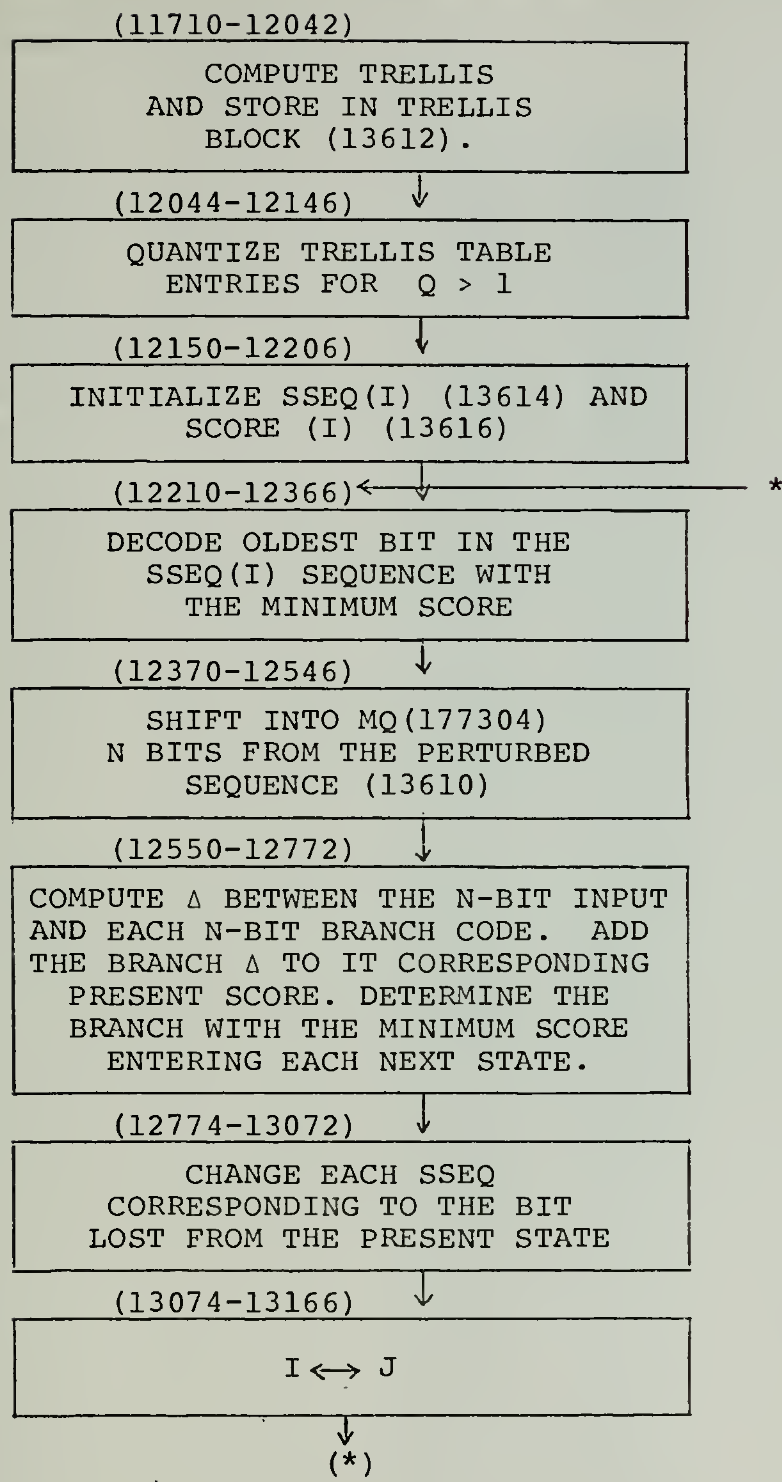
The following computer printout is the encoder implementation just discussed.

```
010710 /016700
010712 /002664
010714 /016701
010716 /002662
010720 /012704
010722 /000020
010724 /016706
010726 /002700
010730 /016716
010732 /002754
010734 /012037
010736 /177304
010740 /005037
010742 /177302
010744 /000406
010746 /013746
010750 /177302
010752 /012037
010754 /177304
010756 /012637
010760 /177302
010762 /012746
010764 /000020
010766 /012737
010770 /000001
010772 /177314
010774 /015746
010776 /002700
011000 /013705
011002 /177302
011004 /016703
011006 /002616
011010 /005105
011012 /012302
011014 /040502
011016 /005046
011020 /016746
011022 /002656
011024 /005726
011026 /006202
011030 /005516
011032 /005346
011034 /001373
011036 /006311
011040 /005726
011042 /006226
011044 /005511
011046 /005304
011050 /001003
011052 /012704
011054 /000020
011056 /005721
011060 /005316
011062 /001353
011064 /005726
011066 /005316
011070 /001336
011072 /005726
011074 /005316
011076 /001323
011100 /005704
011102 /001403
011104 /006311
011106 /005304
011110 /001375
011112 /000000
```

APPENDIX D

Program Listing of Viterbi Decoder

The following flow chart depicts the flow of the Viterbi decoding algorithm implementation. The corresponding machine language program follows the flow chart and the instruction addresses are indicated on top of the respective flow chart block to which they correspond in the subprogram.


(11710-12042)
COMPUTE TRELLIS AND STORE IN TRELLIS BLOCK (13612).
(12044-12146)
QUANTIZE TRELLIS TABLE ENTRIES FOR Q > 1
(12150-12206)
INITIALIZE SSEQ(I) (13614) AND SCORE (I) (13616)
(12210-12366)
*
DECODE OLDEST BIT IN THE SSEQ(I) SEQUENCE WITH THE MINIMUM SCORE
(12370-12546)
SHIFT INTO MQ(177304) N BITS FROM THE PERTURBED SEQUENCE (13610)
(12550-12772)
COMPUTE Δ BETWEEN THE N-BIT INPUT AND EACH N-BIT BRANCH CODE. ADD THE BRANCH Δ TO IT CORRESPONDING PRESENT SCORE. DETERMINE THE BRANCH WITH THE MINIMUM SCORE ENTERING EACH NEXT STATE.
(12774-13072)
CHANGE EACH SSEQ CORRESPONDING TO THE BIT LOST FROM THE PRESENT STATE
(13074-13166)
I ⟷ J
(*)

The following computer printout is the decoder implementation just discussed.

```
011710 /016700
011712 /002014
011714 /016701
011716 /001706
011720 /016702
011722 /001666
011724 /016704
011726 /001700
011730 /005003
011732 /005012
011734 /016714
011736 /001740
011740 /005103
011742 /012105
011744 /040305
011746 /016744
011750 /001730
011752 /005006
011754 /006205
011756 /005506
011760 /005314
011762 /001374
011764 /006312
011766 /006206
011770 /005512
011772 /005724
011774 /005314
011776 /001361
012000 /016701
012002 /001622
012004 /005722
012006 /005103
012010 /026703
012012 /001712
012014 /001407
012016 /020300
012020 /002002
012022 /060003
012024 /000742
012026 /160003
012030 /005203
012032 /000737
012034 /022767
012036 /000001
012040 /001642
012042 /001442
012044 /016700
012046 /001542
012050 /016701
012052 /001650
012054 /016702
012056 /001550
012060 /016703
012062 /001614
012064 /016704
012066 /001614
012070 /005012
012072 /006210
012074 /103403
012076 /006312
012100 /077402
012102 /000403
012104 /006312
012106 /005212
012110 /077403
012112 /005742
012114 /077315
012116 /005722
012120 /016703
012122 /001554
012124 /005010
012126 /016704
012130 /001552
012132 /006310
012134 /077402
012136 /062210
012140 /077306
012142 /005720
012144 /005742
012146 /077136
012150 /016700
012152 /001436
012154 /016701
012156 /001434
012160 /016702
012162 /001432
012164 /016703
012166 /001420
012170 /005021
012172 /005022
012174 /016704
012176 /001532
012200 /005021
012202 /016722
012204 /001514
012206 /077404
```

```
012210 /016701
012212 /001400
012214 /016702
012216 /001376
012220 /016704
012222 /001356
012224 /016767
012226 /001460
012230 /001410
012232 /012767
012234 /000020
012236 /001404
012240 /012337
012242 /177304
012244 /012767
012246 /000020
012250 /001374
012252 /005367
012254 /001462
012256 /002051
012260 /016767
012262 /001444
012264 /001362
012266 /010267
012270 /001360
012272 /012267
012274 /001356
012276 /005367
012300 /001346
012302 /001405
012304 /021267
012306 /001344
012310 /002766
012312 /005722
012314 /000770
012316 /166767
012320 /001422
012322 /001326
012324 /016706
012326 /001322
012330 /006314
012332 /036716
012334 /001404
012336 /100004
012340 /000240
012342 /000240
012344 /000240
012346 /005214
012350 /005367
012352 /001272
012354 /001012
012356 /005724
012360 /005367
012362 /001256
012364 /001003
012366 /000000
012370 /000240
012372 /000240
012374 /012767
012376 /000020
012400 /001244
012402 /010102
012404 /066702
012406 /001334
012410 /005037
012412 /177302
012414 /016706
012416 /001276
012420 /012737
012422 /000001
012424 /177314
012426 /005367
012430 /001212
012432 /001402
012434 /077607
012436 /000414
012440 /012767
012442 /000020
012444 /001176
012446 /013767
012450 /177302
012452 /001156
012454 /012337
012456 /177304
012460 /016737
012462 /001146
012464 /177302
012466 /000762
012470 /016767
012472 /001240
012474 /001160
012476 /010267
012500 /001250
012502 /010167
012504 /001246
012506 /020267
```

012510	/001106
012512	/100010
012514	/066767
012516	/001226
012520	/001230
012522	/066767
012524	/001220
012526	/001224
012530	/000407
012532	/000240
012534	/166767
012536	/001206
012540	/001210
012542	/166767
012544	/001200
012546	/001204
012550	/012767
012552	/000002
012554	/001200
012556	/011267
012560	/001076
012562	/016706
012564	/001112
012566	/013767
012570	/177302
012572	/001044
012574	/012005
012576	/074567
012600	/001036
012602	/005067
012604	/001026
012606	/005167
012610	/001026
012612	/016705
012614	/001066
012616	/016767
012620	/001114
012622	/001012
012624	/046767
012626	/001010
012630	/001004
012632	/066767
012634	/001000
012636	/000774
012640	/006267
012642	/000774
012644	/077503
012646	/077617
012650	/000240
012652	/066767
012654	/000756
012656	/001000
012660	/010267
012662	/000746
012664	/066767
012666	/001040
012670	/000740
012672	/017767
012674	/000734
012676	/000762
012700	/016706
012702	/000774
012704	/013767
012706	/177302
012710	/000726
012712	/012005
012714	/074567
012716	/000720
012720	/005067
012722	/000710
012724	/005167
012726	/000710
012730	/016705
012732	/000750
012734	/016767
012736	/000776
012740	/000674
012742	/046767
012744	/000672
012746	/000666
012750	/066767
012752	/000662
012754	/000656
012756	/006267
012760	/000656
012762	/077503
012764	/077617
012766	/066767
012770	/000642
012772	/000666
012774	/026767
012776	/000660
013000	/000660
013002	/003421
013004	/016705
013006	/000742

```
013010 /016715
013012 /000646
013014 /010167
013016 /000644
013020 /066767
013022 /000704
013024 /000636
013026 /016705
013030 /000632
013032 /016706
013034 /000716
013036 /011516
013040 /006316
013042 /005216
013044 /000410
013046 /016705
013050 /000700
013052 /016706
013054 /000676
013056 /016715
013060 /000576
013062 /011116
013064 /006316
013066 /016705
013070 /000660
013072 /022526
013074 /010567
013076 /000652
013100 /010667
013102 /000650
013104 /005367
013106 /000646
013110 /001222
013112 /022122
013114 /000240
013116 /005367
013120 /000534
013122 /001212
013124 /016700
013126 /000462
013130 /020267
013132 /000464
013134 /100406
013136 /016701
013140 /000452
013142 /016702
013144 /000450
013146 /162707
013150 /000700
013152 /016701
013154 /000442
013156 /016702
013160 /000440
013162 /162707
013164 /000714
013166 /000000
```

APPENDIX E

Program Listing of Supplementary Subprograms

1. Initialization subprogram (10174-10706)

This subprogram enters the necessary addresses and constants into those locations noted in Appendix A by *. The computer listing below is the initialization subprogram.

```
010174 /010700
010176 /012767
010200 /000002
010202 /003474
010204 /012767
010206 /000003
010210 /003470
010212 /012767
010214 /000001
010216 /003464
010220 /012767
010222 /000020
010224 /003460
010226 /012767
010230 /000001
010232 /003456
010234 /012767
010236 /000000
010240 /003452
010242 /010701
010244 /062701
010246 /003514
010250 /010167
010252 /003352
010254 /062701
010256 /000406
010260 /010167
010262 /003344
010264 /062700
010266 /003402
010270 /010001
010272 /062700
010274 /000600
010276 /010021
010300 /066700
010302 /003404
010304 /010021
010306 /016737
010310 /003366
010312 /177304
010314 /016737
010316 /003370
010320 /177306
010322 /013767
010324 /177304
010326 /003420
010330 /063700
010332 /177304
010334 /010021
010336 /016737
010340 /003342
010342 /177306
010344 /013767
010346 /177304
010350 /003346
010352 /063700
010354 /177304
010356 /010021
010360 /063700
010362 /177304
```

```
010364 /010021
010366 /063700
010370 /177304
010372 /010021
010374 /005002
010376 /005202
010400 /016703
010402 /003276
010404 /006302
010406 /077302
010410 /010267
010412 /003310
010414 /006302
010416 /060200
010420 /010021
010422 /006202
010424 /005302
010426 /010267
010430 /003274
010432 /005202
010434 /012703
010436 /000003
010440 /060200
010442 /010021
010444 /077303
010446 /006202
010450 /010267
010452 /003254
010454 /005302
010456 /010267
010460 /003250
010462 /005202
010464 /006202
010466 /010267
010470 /003242
010472 /012737
010474 /000004
010476 /177304
010500 /016737
010502 /003176
010504 /177306
010506 /013767
010510 /177304
010512 /003206
010514 /016737
010516 /003164
010520 /177304
010522 /016737
010524 /003152
010526 /177306
010530 /013767
010532 /177304
010534 /003160
010536 /016702
010540 /003142
010542 /005003
010544 /005203
010546 /006303
010550 /077202
010552 /005303
010554 /010367
010556 /003156
010560 /005203
010562 /006203
010564 /010367
010566 /003106
010570 /016702
010572 /003106
010574 /005302
010576 /066702
010600 /003104
010602 /010267
010604 /003132
010606 /016702
010610 /003074
010612 /005302
010614 /005003
010616 /005203
010620 /006303
010622 /077202
010624 /010367
010626 /003112
010630 /016702
010632 /002764
010634 /166702
010636 /002756
010640 /010267
010642 /003100
010644 /016702
010646 /002752
010650 /166702
010652 /002742
010654 /010267
010656 /003066
010660 /010700
010662 /062700
010664 /001364
010666 /010067
010670 /002304
010672 /006267
010674 /003012
010676 /006267
010700 /003016
010702 /006267
010704 /003042
010706 /000000
```

2. Message Input Subprogram (10000-10144)

This subprogram was used to indicate that an alphanumeric symbol typed at the Keyboard was entered into core (echo). The symbols are stored in core in an ASC II (7-bit) code representation. The Keyboard symbol, @ , is used to terminate message entry and the number of 8-bit computer bytes used is stored in location (13710 Appendix A). The following computer printout is a listing of the ASC II message input subprogram.

```
010000 /010700        010064 /000012
010002 /062700        010066 /105737
010004 /004376        010070 /177564
010006 /005002        010072 /100375
010010 /105737        010074 /112737
010012 /177560        010076 /000200
010014 /100375        010100 /177566
010016 /113710        010102 /077107
010020 /177562        010104 /105737
010022 /122710        010106 /177564
010024 /000300        010110 /100375
010026 /001435        010112 /112737
010030 /105737        010114 /000212
010032 /177564        010116 /177566
010034 /100375        010120 /000733
010036 /112037        010122 /012703
010040 /177566        010124 /000010
010042 /005202        010126 /105020
010044 /123727        010130 /005202
010046 /177562        010132 /077303
010050 /000215        010134 /006202
010052 /001356        010136 /006302
010054 /112740        010140 /010267
010056 /000240        010142 /003544
010060 /105720        010144 /000000
010062 /012701        *
```

3. Message Output Subprogram (13200-13364)

This subprogram is basically the same as that in section 2 of this appendix. However, besides typing what is in core starting at a location specified at (13602), the number of errors (bit differences) are determined between the input message and the decoded out message. This message error value is stored at (13672) (Appendix A). The following computer printout is a listing of the ASC II (7-bit) code output subprogram.

```
013200 /016700        013274 /100375
013202 /000376        013276 /112737
013204 /012701        013300 /000012
013206 /000040        013302 /177566
013210 /111037        013304 /105737
013212 /177566        013306 /177564
013214 /122720        013310 /100375
013216 /000000        013312 /000734
013220 /001435        013314 /000240
013222 /105737        013316 /000240
013224 /177564        013320 /016700
013226 /100375        013322 /000254
013230 /005301        013324 /016701
013232 /100366        013326 /000252
013234 /122710        013330 /005002
013236 /000240        013332 /011003
013240 /001363        013334 /074311
013242 /112737        013336 /012704
013244 /000015        013340 /000020
013246 /177566        013342 /006211
013250 /012702        013344 /005502
013252 /000012        013346 /077403
013254 /105737        013350 /022021
013256 /177564        013352 /020067
013260 /100375        013354 /000224
013262 /105037        013356 /001365
013264 /177566        013360 /010267
013266 /077206        013362 /000306
013270 /105737        013364 /000000
013272 /177564        *
```

4. Analysis Subprograms (13250-13544)

From address (13250) to address (13376) a short subprogram is listed that is used to step through well defined encoders of rate 1/2, 1/3, and 1/4. In the last addresses (13400-13544) a program is listed which determines the number of error bits in 16-quantized bits and stores the results so that a distribution of errors may be plotted as in figures 17-23. (See next page.)

```
013250 /005267
013252 /000512
013254 /026727
013256 /000506
013260 /000010
013262 /001030
013264 /012767
013266 /000004
013270 /000474
013272 /005267
013274 /000466
013276 /026727
013300 /000462
013302 /000010
013304 /001017
013306 /012767
013310 /000004
013312 /000450
013314 /005267
013316 /000442
013320 /026727
013322 /000436
013324 /000020
013326 /001006
013330 /012767
013332 /000010
013334 /000424
013336 /062767
013340 /000002
013342 /000414
013344 /016777
013346 /000322
013350 /000416
013352 /062767
013354 /000002
013356 /000410
013360 /026727
013362 /000374
013364 /000021
013366 /001402
013370 /000137
013372 /010000
013374 /000137
013376 /001172
013400 /016700
013402 /000202
013404 /016701
013406 /000326
013410 /006201
013412 /000240
013414 /012767
013416 /000020
013420 /000214
013422 /012704
013424 /014100
013426 /012705
013430 /000020
013432 /000240
013434 /000240
013436 /000240
013440 /000240
013442 /005006
013444 /016703
013446 /000234
013450 /005002
013452 /006302
013454 /006310
013456 /005502
013460 /005367
013462 /000152
013464 /001407
013466 /077307
013470 /020201
013472 /003401
013474 /005206
013476 /005305
013500 /001414
013502 /000760
013504 /005720
013506 /026700
013510 /000076
013512 /002003
013514 /000000
013516 /000240
013520 /000240
013522 /012767
013524 /000020
013526 /000106
013530 /000756
013532 /006306
013534 /060604
013536 /005214
013540 /000240
013542 /000727
013544 /000000
```

APPENDIX F

SAMPLE RUN

THE COMPUTER LISTING BELOW THIS PARAGRAPH IS A DEMONSTRATION OF THE PROCEDURE REQUIRED TO USE THE ENTIRE PROGRAM PROPERLY. THE PROGRAM IS STORED ON A DISK UNDER THE NAME VSAC.SAV. THE CONVOLUTIONAL CODE USED IS A RATE 1/2 (K=3) CODE. OTHER INPUTS ARE Q=1, DCL=16 (20 IN BASE 8), AND THE GENERATOR SEQUENCES OF 3 AND 4 (BASE 8).

```
^C

.GET VSAC.SAV

.START 1172
ODT  V01-01
*
*10200/000002
*10206/000003
*10214/000001
*10222/000020
*10230/000001
*10236/000001
*
*12336/100004
*
*13760/000003
*13762/000004
*
*^C
.START 10000
```

The paragraph below is the input message for the sample run.

```
     THIS IS A SAMPLE RUN OF A THESIS COMPUTER
PROGRAM USED TO EVALUATE THE PERFORMANCE OF RATE
1/N CONVOLUTIONAL CODES, OVER A SIMULATED NOISY
CHANNEL, WITH A VITERBI DECODER.  THE NOISE PARAMETERS
FOR THIS RUN (CORE LOCATIONS 10230 AND 10236) ARE
THE SAME AS WERE USED TO OBTAIN THE DISTRIBUTION
IN FIGURE 20 OF THE THESIS TEXT.  THIS MESSAGE WILL
NOW BE TERMINATED BY THE 'AT' CHARACTER ON THE PANEL.
```

When the message above is encoded (as given above) and the noise is added, the resulting decoded message is:

```
     THIS IS!A"SQMPLE RUN OF E THESIS
COMPUTR PROGBAM USED \O EVALUATE
THE ERFNRUANCE@OF RATE 1/N CONVOLUTIONAL!@CDES,
OVMR AFIMULBTEE NOISY PANNEL  ITH(@
VITERBI DECOLDR.  UHE NOISE PARAMETERS
FOR TPIS UN (COU LOCATIONS 10230
AND 102361 ARU $E SAUA AS TEVE USED"TO
OBTAIN THE @ISTRIBUTION IN@FJURE
21 OF THE THEIP TEXD.  THIS MESSAGD
WILL NIW BE TERMINATED BY"THM 'AX'
CHBRCTER ON THM QBNE@.
```

When no coding is applied to the input message and the same noise is added the received message is:

```
0LAF@NINBRR@B0QJ@ME@VUN"MD00(TICK[0COMXU
DR ROAM(URM@$D_(MTCM]AUE D@A0QDRFOEAFKD"F
SKUI"A!RMVAVRM"EGCOER>0@\@D!NNASD"TCRQI
UUGRS0NKV                                  ← *
PJIS ZUO!HCORE$LBITIOJ[10 :7"ED"943;V9$
R@EBIOU$IB$GRM0ECUD@TOOM@VEIN!DUDSTPHCWUI
YD@E@$BB0IRU!B0TE$NOG(E TGR
 BJRIKFAR"_NDJM!XNEL. @
```

* Improper carriage return received.

LIST OF REFERENCES

1. Computer Sciences Group, Error Protection Manual, Falls Church, Va. November 1972.

2. Elias, P., "Coding for Noisy Channels," IRE Convention Record, part IV, pp. 37-46, 1955.

3. Forney, G.D., "The Viterbi Algorithm," Proceedings of the IEEE, Vol. 61, No. 3, pp. 268-278, March 1973.

4. Huth, G. K., and Weber, C.L., Notes on Convolutional Codes, course notes presented at USC, March 1973.

5. Peterson, W.W., and Weldon, E.J., Jr., Error Correcting Codes, 2nd Edition MIT Press, 1972.

6. Rice, S. O., "Communication in the Presense of Noise-Probability of Error for Two Encoding Schemes," Bell System Technical Journal, Vol. 29, pp. 60-93, January 1950.

7. Shannon, C.E., "The Mathematical Theory of Communication," Bell System Technical Journal, Vol. 27, pp. 379-423 and 623-656, July and October 1948.

8. Viterbi, A.J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," IEEE Transactions on Information Theory, Vol. I T-13, No. 2, pp. 260-269, April 1967.

9. Viterbi, A.J., "Convolutional Codes and Their Performance in Communication Systems," IEEE Transactions on Communications Technology, Vol. COM-19, No. 5, pp. 751-772, October 1971.

INITIAL DISTRIBUTION LIST

	No. Copies
Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
Department Chairman, Code 52 Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	2
Professor George H. Marmont, Code 52Ma (Thesis Advisor Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	10
Captain James H. Haney, USMC (Student) 1187 Barbara Court Seaside, California 93940	1
LtCol. Robert W. Burton, USAF, Code 52Zn Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
Professor Michael Powers, Code 52Pw Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
Mr. Ray Kelly NELC, Code 3200 271 Catalina Blvd. San Diego, California 92152	1